



**RED HAT®
TRAINING**



Comprehensive, hands-on training that solves real world problems

Red Hat OpenShift Administration II: High Availability

Student Workbook (ROLE)

**RED HAT
OPENShift
ADMINISTRATION II:
HIGH AVAILABILITY**

Openshift Container Platform 3.6 DO380
Red Hat OpenShift Administration II: High Availability
Edition 2 20180213 20180213

Authors: Michael Jarrett, Razique Mahroua, Ricardo Taniguchi, Victor Costea
Editor: Dave Sacco

Copyright © 2017 Red Hat, Inc.

The contents of this course and all its modules and related materials, including handouts to audience members, are Copyright © 2017 Red Hat, Inc.

No part of this publication may be stored in a retrieval system, transmitted or reproduced in any way, including, but not limited to, photocopy, photograph, magnetic, electronic or other record, without the prior written permission of Red Hat, Inc.

This instructional program, including all material provided herein, is supplied without any guarantees from Red Hat, Inc. Red Hat, Inc. assumes no liability for damages or legal action arising from the use or misuse of contents or details contained herein.

If you believe Red Hat training materials are being used, copied, or otherwise improperly distributed please e-mail training@redhat.com or phone toll-free (USA) +1 (866) 626-2994 or +1 (919) 754-3700.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, Hibernate, Fedora, the Infinity Logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux® is the registered trademark of Linus Torvalds in the United States and other countries.

Java® is a registered trademark of Oracle and/or its affiliates.

XFS® is a registered trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

The OpenStack® Word Mark and OpenStack Logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Document Conventions	vii
Notes and Warnings	vii
Introduction	ix
Red Hat OpenShift Administration II	ix
Orientation to the Classroom Environment	x
Internationalization	xiii
1. Designing a Highly Available Cluster	1
Describing Highly Available OpenShift Architecture	2
Quiz: Describing Highly Available OpenShift Architecture	7
Determining Networking and Sizing Requirements	9
Guided Exercise: Determining Networking and Sizing Requirements	14
Determining Load Balancing and Pod Distribution Schemes	16
Quiz: Determining Load Balancing and Pod Distribution Schemes	18
Lab: Designing a Highly Available Cluster	20
Summary	27
2. Preparing to Install an HA Cluster	29
Describing the Advanced Installation Method	30
Quiz: Describing the Advanced Installation Method	35
Building the Ansible Inventory	37
Guided Exercise: Building the Ansible Inventory	42
Configuring the Integrated Registry	45
Guided Exercise: Configuring the Integrated Registry	51
Preparing a Disconnected Environment	55
Guided Exercise: Preparing a Disconnected Environment	61
Lab: Preparing to Install an HA Cluster	68
Summary	79
3. Configuring OpenShift to use Custom Certificates	81
Describing an OpenShift Certificate Management Strategy	82
Quiz: Describing OpenShift Certificate Management Strategies	86
Configuring Custom Certificates	88
Guided Exercise: Configuring OpenShift to use an Intermediate Certificate Authority (CA)	91
Lab: Configuring OpenShift to Use Custom Certificates	97
Summary	101
4. Building a Highly Available Cluster	103
Provisioning Servers	104
Guided Exercise: Provisioning Servers	111
Executing Pre-installation Tasks	117
Guided Exercise: Executing Pre-installation Tasks	123
Running the Advanced Installation Method	129
Guided Exercise: Using the Advanced Installation Method	140
Lab: Building a Highly Available Cluster	149
Summary	159
5. Provisioning Persistent Storage	161
Describing Storage Providers	162
Quiz: Describing Storage Providers	169
Configuring Storage Providers	171
Guided Exercise: Configuring Storage Providers	175

Configuring and Testing Red Hat Gluster Container-Native Storage	178
Guided Exercise: Configuring and Testing Red Hat Gluster Storage CNS	181
Lab: Provisioning Persistent Storage	186
Summary	196
6. Enabling Log Aggregation	197
Describing Log Aggregation	198
Quiz: Describing Log Aggregation	205
Enabling Log Aggregation in an HA Cluster	209
Guided Exercise: Enabling Log Aggregation in an HA Cluster	216
Lab: Enabling Log Aggregation	224
Summary	230
7. Maintaining an OpenShift Cluster	231
Performing Diagnostics Checks	232
Guided Exercise: Performing Diagnostics Checks	239
Backing up and Restoring Key Data Stores	244
Guided Exercise: Backing Up and Restoring Key Data Stores	247
Cleaning up the Cluster	253
Guided Exercise: Cleaning up the Cluster	258
Adding and Removing Cluster Nodes	261
Guided Exercise: Adding and Removing Cluster Nodes	264
Quiz: Maintaining An OpenShift Cluster	268
Summary	271
8. Managing System Resources	273
Configuring Admission Controllers	274
Guided Exercise: Configuring Admission Controllers	282
Configuring OpenShift for Overcommitment and Idling Services	286
Guided Exercise: Idling a Service	289
Managing Operating System and OpenShift Resources	292
Guided Exercise: Managing Operating System and OpenShift Resources	298
Lab: Managing System Resources	305
Summary	314
9. Configuring Security Providers and Advanced Security Options	315
Describing OpenShift Container Platform Security Providers	316
Quiz: Describing OpenShift Container Platform Security Providers	328
Configuring External Security Providers	330
Guided Exercise: Configuring the LDAP Security Provider	334
Configuring User Security	339
Guided Exercise: Configuring User Security	349
Lab: Configuring Security Providers	354
Summary	357
10. Configuring Networking Options	359
Describing Load Balancing, Failover, and Cluster Connectivity Settings	360
Quiz: Describing Load Balancing, Failover, and Cluster Connectivity Settings	369
Configuring the Multitenant Software-defined Networking Provider	371
Guided Exercise: Configuring the Multitenant Software-defined Networking Provider	378
Describing iptables Management	381
Quiz: Describing OpenShift Container Platform's iptables Management	384
Lab: Configuring Networking Options	386
Summary	392

Document Conventions

Notes and Warnings



Note

"Notes" are tips, shortcuts or alternative approaches to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

"Important" boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring a box labeled "Important" will not cause data loss, but may cause irritation and frustration.



Warning

"Warnings" should not be ignored. Ignoring warnings will most likely cause data loss.



References

"References" describe where to find external documentation relevant to a subject.

Introduction

Red Hat OpenShift Administration II

Red Hat OpenShift Administration II (DO380) prepares system administrators to install, configure, and manage large OpenShift clusters using Red Hat OpenShift Container Platform.

This course explores OpenShift administration topics such as:

- Advanced installation method
- External security providers
- Dynamic storage
- Certificate management

In this course, you will use the advanced installation method to build an HA cluster, which you will use to learn various skills in the hands-on labs.

Objectives

- Design and build an OpenShift HA cluster.
- Manage certificates and security providers for the cluster.
- Provision storage using dynamic storage providers.
- Enable log aggregation.
- Maintain a cluster and the resources it consumes.

Audience

Linux system administrators interested in deploying and managing a large-scale OpenShift Container Platform deployment in their data centers.

Prerequisites

The prerequisites for this class are as follows:

- RHCSA certification or equivalent experience.
- *Introduction to Containers, Kubernetes, and Red Hat OpenShift* (DO180) attendance or equivalent experience with containers, Kubernetes, and OpenShift.
- *Red Hat OpenShift Administration I* (DO280) or equivalent experience with OpenShift.

Red Hat also recommends that students have a *Certificate of Expertise in Platform-as-a-Service* (EX280).

Orientation to the Classroom Environment

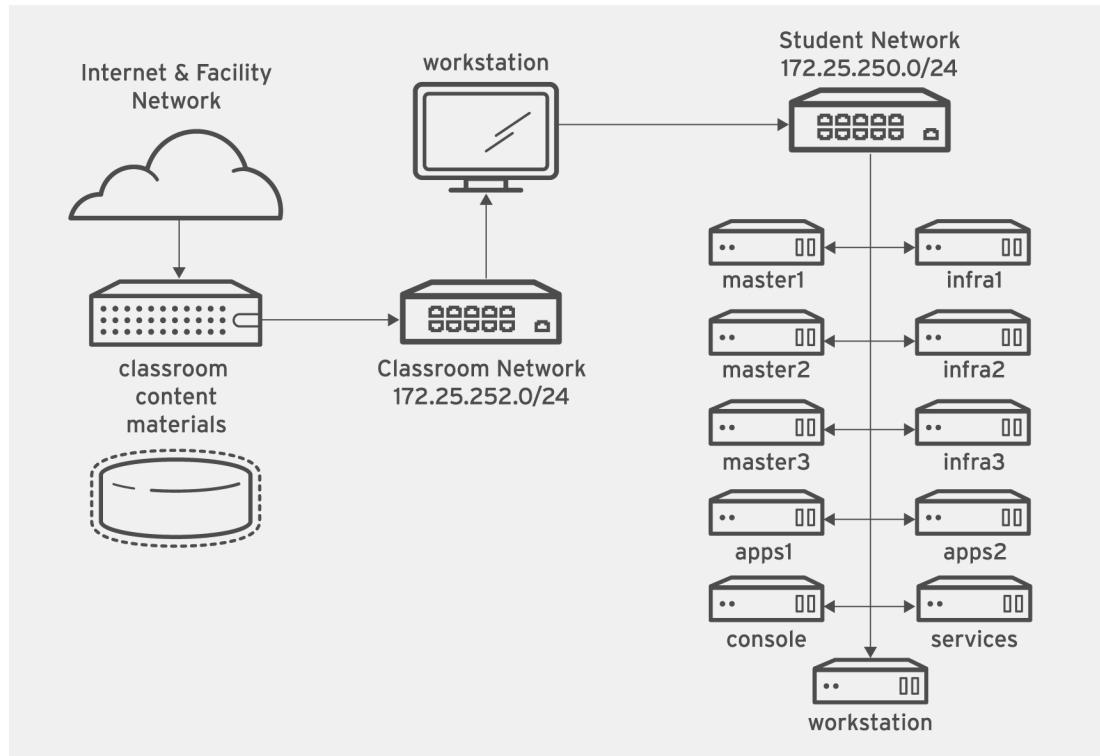


Figure 0.1: Classroom environment

In this course, the main computer system used for hands-on learning activities is **workstation**. Ten other machines will also be used by students for these activities. These are **master1**, **master2**, **master3**, **infra1**, **infra2**, **infra3**, **console**, **apps1**, **apps2**, and **services**. All eleven of these systems are in the **lab.example.com** DNS domain.

All student computer systems have a standard user account, **student**, which has the password **student**. The **root** password on all student systems is **redhat**.

Classroom Machines

Machine name	IP addresses	Role
workstation.lab.example.com	172.25.250.254	Graphical workstation used for system administration
master1.lab.example.com	172.25.250.11	OpenShift master node 1
master2.lab.example.com	172.25.250.12	OpenShift master node 2
master3.lab.example.com	172.25.250.13	OpenShift master node 3
infra1.lab.example.com	172.25.250.21	OpenShift infrastructure node 1
infra2.lab.example.com	172.25.250.22	OpenShift infrastructure node 2
infra3.lab.example.com	172.25.250.23	OpenShift infrastructure node 3
apps1.lab.example.com	172.25.250.31	OpenShift application node 1
apps2.lab.example.com	172.25.250.32	OpenShift application node 2

Machine name	IP addresses	Role
services.lab.example.com / registry.lab.example.com	172.25.250.40	Host with services needed by OpenShift
console.lab.example.com	172.25.250.10	Load balancer, Ansible Control, and NFS shares

One additional function of **workstation** is that it acts as a router between the network that connects the student machines and the classroom network. If **workstation** is down, other student machines will only be able to access systems on the student network.

There are several systems in the classroom that provide supporting services. Two servers, **content.example.com** and **materials.example.com**, are sources for software and lab materials used in hands-on activities. Information on how to use these servers is provided in the instructions for those activities.

Exercise Prerequisites and Starting Over

This OpenShift Administration course configures a High Availability environment, chapter by chapter. It is expected that students perform lab exercises in chapter and section order throughout this course.

Chapters 1 through 4 perform preliminary tasks necessary for building a working High Availability cluster. The chapter 4 final lab results in a working cluster required for the remainder of the course. Mindful adherence to subsequent lab instructions results in successful lab completion. Unsuccessful results are to be diagnosed and fixed before continuing. If a working environment becomes unfixable, or starting over is necessary for any reason, reset the course environment to the initial course state for all VMs, and reperform all lab exercises.

The **lab build-cluster** command is provided to automate rebuilding the cluster. Running this command, after resetting all VMs, performs all necessary exercise tasks up through chapter 4, resulting in a working cluster. Exercises in chapters 5 and above require a working cluster.



Important

Resetting the working environment, and starting again, bypasses any unresolvable problem, but should not be a decision taken lightly. Resetting and rebuilding the cluster, even when automated by **lab build-cluster**, takes between 30 to 90 minutes, depending on hardware performance characteristics. If the course environment is reset, performing the labs in Chapters 5, 6, 7, 8, 9 and 10 require creating a working cluster first. A working cluster is created by reperforming all chapter 1 through chapter 4 exercises, or by running this equivalent command from **workstation**:

```
[student@workstation ~]$ lab build-cluster setup
```

Controlling Your Station

The top of the console describes the state of your machine.

Machine States

State	Description
none	Your machine has not yet been started. When started, your machine will boot into a newly initialized state (the desk will have been reset).

State	Description
starting	Your machine is in the process of booting.
running	Your machine is running and available (or, when booting, soon will be.)
stopping	Your machine is in the process of shutting down.
stopped	Your machine is completely shut down. Upon starting, your machine will boot into the same state as when it was shut down (the disk will have been preserved).
impaired	A network connection to your machine cannot be made. Typically this state is reached when a student has corrupted networking or firewall rules. If the condition persists after a machine reset, or is intermittent, please open a support case.

Depending on the state of your machine, a selection of the following actions will be available to you.

Machine Actions

Action	Description
Start Station	Start (power on) the machine.
Stop Station	Stop (power off) the machine, preserving the contents of its disk.
Reset Station	Stop (power off) the machine, resetting the disk to its initial state. Caution: Any work generated on the disk will be lost.
Refresh	Re-probe the machine state.
Increase Timer	Add 15 minutes to the timer for each click.

The Station Timer

Your Red Hat Online Learning enrollment entitles you to a certain amount of computer time. In order to help you conserve your time, the machines have an associated timer, which is initialized to 60 minutes when your machine is started.

The timer operates as a "dead man's switch," which decrements as your machine is running. If the timer is approaching zero, you can choose to increase the remaining time.

Internationalization

Language Support

Red Hat Enterprise Linux 7 officially supports 22 languages: English, Assamese, Bengali, Chinese (Simplified), Chinese (Traditional), French, German, Gujarati, Hindi, Italian, Japanese, Kannada, Korean, Malayalam, Marathi, Odia, Portuguese (Brazilian), Punjabi, Russian, Spanish, Tamil, and Telugu.

Per-user Language Selection

Users may prefer to use a different language for their desktop environment than the system-wide default. They may also want to set their account to use a different keyboard layout or input method.

Language Settings

In the GNOME desktop environment, the user may be prompted to set their preferred language and input method on first login. If not, then the easiest way for an individual user to adjust their preferred language and input method settings is to use the **Region & Language** application. Run the command **gnome-control-center region**, or from the top bar, select **(User) > Settings**. In the window that opens, select **Region & Language**. The user can click the **Language** box and select their preferred language from the list that appears. This will also update the **Formats** setting to the default for that language. The next time the user logs in, these changes will take full effect.

These settings affect the GNOME desktop environment and any applications, including **gnome-terminal**, started inside it. However, they do not apply to that account if accessed through an **ssh** login from a remote system or a local text console (such as **tty2**).



Note

A user can make their shell environment use the same **LANG** setting as their graphical environment, even when they log in through a text console or over **ssh**. One way to do this is to place code similar to the following in the user's **~/.bashrc** file. This example code will set the language used on a text login to match the one currently set for the user's GNOME desktop environment:

```
i=$(grep 'Language=' /var/lib/AccountService/users/${USER} \
    | sed 's/Language=//')
if [ "$i" != "" ]; then
    export LANG=$i
fi
```

Japanese, Korean, Chinese, or other languages with a non-Latin character set may not display properly on local text consoles.

Individual commands can be made to use another language by setting the **LANG** variable on the command line:

```
[user@host ~]$ LANG=fr_FR.utf8 date
```

jeu. avril 24 17:55:01 CDT 2014

Subsequent commands will revert to using the system's default language for output. The **locale** command can be used to check the current value of **LANG** and other related environment variables.

Input Method Settings

GNOME 3 in Red Hat Enterprise Linux 7 automatically uses the **IBus** input method selection system, which makes it easy to change keyboard layouts and input methods quickly.

The **Region & Language** application can also be used to enable alternative input methods. In the **Region & Language** application's window, the **Input Sources** box shows what input methods are currently available. By default, **English (US)** may be the only available method. Highlight **English (US)** and click the **keyboard** icon to see the current keyboard layout.

To add another input method, click the **+** button at the bottom left of the **Input Sources** window. An **Add an Input Source** window will open. Select your language, and then your preferred input method or keyboard layout.

Once more than one input method is configured, the user can switch between them quickly by typing **Super+Space** (sometimes called **Windows+Space**). A *status indicator* will also appear in the GNOME top bar, which has two functions: It indicates which input method is active, and acts as a menu that can be used to switch between input methods or select advanced features of more complex input methods.

Some of the methods are marked with gears, which indicate that those methods have advanced configuration options and capabilities. For example, the Japanese **Japanese (Kana Kanji)** input method allows the user to pre-edit text in Latin and use **Down Arrow** and **Up Arrow** keys to select the correct characters to use.

US English speakers may find also this useful. For example, under **English (United States)** is the keyboard layout **English (international AltGr dead keys)**, which treats **AltGr** (or the right **Alt**) on a PC 104/105-key keyboard as a "secondary-shift" modifier key and dead key activation key for typing additional characters. There are also Dvorak and other alternative layouts available.



Note

Any Unicode character can be entered in the GNOME desktop environment if the user knows the character's Unicode code point, by typing **Ctrl+Shift+U**, followed by the code point. After **Ctrl+Shift+U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.

For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl+Shift+U**, then **03bb**, then **Enter**.

System-wide Default Language Settings

The system's default language is set to US English, using the UTF-8 encoding of Unicode as its character set (**en_US.utf8**), but this can be changed during or after installation.

From the command line, *root* can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it will display the current system-wide locale settings.

To set the system-wide language, run the command **localectl set-locale LANG=locale**, where *locale* is the appropriate **\$LANG** from the "Language Codes Reference" table in this chapter. The change will take effect for users on their next login, and is stored in **/etc/locale.conf**.

```
[root@host ~]# localectl set-locale LANG=fr_FR.utf8
```

In GNOME, an administrative user can change this setting from **Region & Language** and clicking the **Login Screen** button at the upper-right corner of the window. Changing the **Language** of the login screen will also adjust the system-wide default language setting stored in the **/etc/locale.conf** configuration file.



Important

Local text consoles such as **tty2** are more limited in the fonts that they can display than **gnome-terminal** and **ssh** sessions. For example, Japanese, Korean, and Chinese characters may not display as expected on a local text console. For this reason, it may make sense to use English or another language with a Latin character set for the system's text console.

Likewise, local text consoles are more limited in the input methods they support, and this is managed separately from the graphical desktop environment. The available global input settings can be configured through **localectl** for both local text virtual consoles and the X11 graphical environment. See the **localectl(1)**, **kbd(4)**, and **vconsole.conf(5)** man pages for more information.

Language Packs

When using non-English languages, you may want to install additional "language packs" to provide additional translations, dictionaries, and so forth. To view the list of available langpacks, run **yum langavailable**. To view the list of langpacks currently installed on the system, run **yum langlist**. To add an additional langpack to the system, run **yum langinstall code**, where *code* is the code in square brackets after the language name in the output of **yum langavailable**.



References

locale(7), **localectl(1)**, **kbd(4)**, **locale.conf(5)**, **vconsole.conf(5)**, **unicode(7)**, **utf-8(7)**, and **yum-langpacks(8)** man pages

Conversions between the names of the graphical desktop environment's X11 layouts and their names in **localectl** can be found in the file **/usr/share/X11/xkb/rules/base.lst**.

Language Codes Reference

Language Codes

Language	\$LANG value
English (US)	en_US.utf8
Assamese	as_IN.utf8
Bengali	bn_IN.utf8
Chinese (Simplified)	zh_CN.utf8
Chinese (Traditional)	zh_TW.utf8
French	fr_FR.utf8
German	de_DE.utf8
Gujarati	gu_IN.utf8
Hindi	hi_IN.utf8
Italian	it_IT.utf8
Japanese	ja_JP.utf8
Kannada	kn_IN.utf8
Korean	ko_KR.utf8
Malayalam	ml_IN.utf8
Marathi	mr_IN.utf8
Odia	or_IN.utf8
Portuguese (Brazilian)	pt_BR.utf8
Punjabi	pa_IN.utf8
Russian	ru_RU.utf8
Spanish	es_ES.utf8
Tamil	ta_IN.utf8
Telugu	te_IN.utf8



CHAPTER 1

DESIGNING A HIGHLY AVAILABLE CLUSTER

Overview	
Goal	Design an OpenShift cluster that supports high availability.
Objectives	<ul style="list-style-type: none">Describe the features and architecture of a highly available (HA) OpenShift cluster.Determine the networking and sizing requirements for an HA cluster.Determine the load balancing and pod distribution schemes for an HA cluster.
Sections	<ul style="list-style-type: none">Describing Highly Available OpenShift Architecture (and Quiz)Determining Networking and Sizing Requirements (and Guided Exercise)Determining Load Balancing and Pod Distribution Schemes (and Quiz)
Lab	Designing a Highly Available Cluster

Describing Highly Available OpenShift Architecture

Objective

After completing this section, students should be able to describe the features and architecture of a highly available (HA) OpenShift cluster.

OpenShift Non-HA Cluster

Within OpenShift Container Platform, Kubernetes manages containerized applications across a set of containers or hosts and provides mechanisms for deployment, maintenance, and application scaling. The Docker service packages, instantiates, and runs containerized applications.

A non-HA OpenShift cluster consists of a master server and a set of application nodes.

Node Servers

A node provides the runtime environments for containers. Each node in an OpenShift cluster has the required services and they are managed by the master. Nodes also have the required services to run pods, including the Docker service, a kubelet, and a service proxy.

OpenShift Container Platform creates nodes from a cloud provider, physical systems, or virtual systems. Kubernetes interacts with node objects, which are a representation of those nodes. The master uses the information from node objects to validate nodes with health checks. A node is ignored until it passes the health checks, and the master continues checking nodes until they are valid.

Master Servers

The master server is the host or hosts that contain the master components, including the API server, controller manager server, and Etcd data store. The master manages nodes in its OpenShift cluster and schedules pods to run on nodes. The following table lists the components of a non-HA master node:

Component	Description
API server	The OpenShift API server validates and configures the data for pods, services, and replication controllers. It also assigns pods to nodes and synchronizes pod information with service configuration. The API server can be run as a standalone process.
etcd	etcd stores the persistent master state while other components watch etcd for changes to bring themselves into the desired state. etcd can be optionally configured for high availability, typically deployed with 2n+1 peer services.
Controller manager server	The controller manager server watches Etcd for changes to replication controller objects and then uses the API to enforce the desired state. The controller manager server can be run as a standalone process. Several such processes create a cluster with one active leader at a time.

OpenShift Cluster Architecture

An OpenShift cluster is a set of node servers that run containers and are centrally managed by a master server. A server can act as both a master and a node, but those roles are usually segregated for increased stability.

The following diagram describes the basic architecture of an OpenShift cluster:

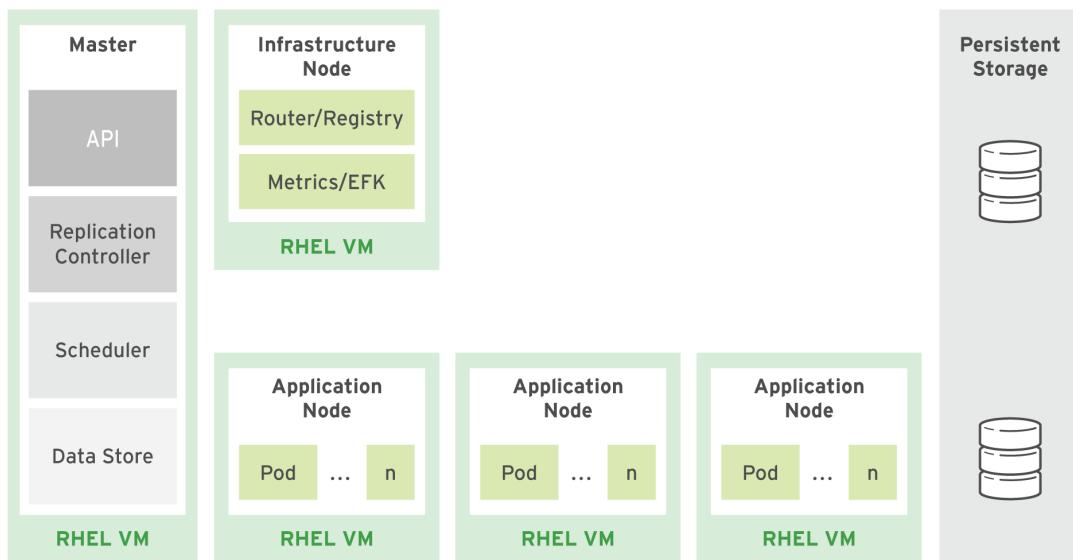


Figure 1.1: OpenShift Container Platform cluster architecture

The major components in the above diagram include:

- A master server
- An infrastructure node
- Three application nodes
- Persistent storage

OpenShift HA Cluster

When using a single master configuration, the availability of running applications remains if the master or any of its services fail. However, failure of master services reduces the ability of the system to respond to application failures or the creation of new applications. Optionally, master servers can be configured for high availability (HA) to ensure that the cluster has no single point of failure.

HA Configuration Types

The advanced installation method provides specific examples using either the native or pacemaker HA method, configuring *HAProxy* or *Pacemaker*, respectively.



Note

The advanced installation method automatically configures the native HA method using HAProxy when three or more master servers are requested.

Using the **native** HA method with HAProxy, the master components will have the following availability:

Role	Style	Notes
API Server	Active-active	Managed by HAProxy.
etcd	Active-active	Fully redundant deployment with load balancing.
Controller Manager Server	Active-passive	One instance is selected as a cluster leader at a time.
HAProxy	Active-passive	Balances load between API master endpoints.

The availability matrix changes slightly when using the Pacemaker method.

Role	Style	Notes
Master Server	Active-passive	One active at a time, managed by Pacemaker.
etcd	Active-active	Fully redundant deployment with load balancing.
Pacemaker	Active-active	Fully redundant deployment.
Virtual IP (VIP)	Active-passive	One active at a time, managed by Pacemaker.

The following diagram describes the architecture of an OpenShift HA cluster, including:

- Enterprise load balancer
- Three master servers
- Three infrastructure servers
- Three applications nodes
- Shared storage backed by persistent storage volumes

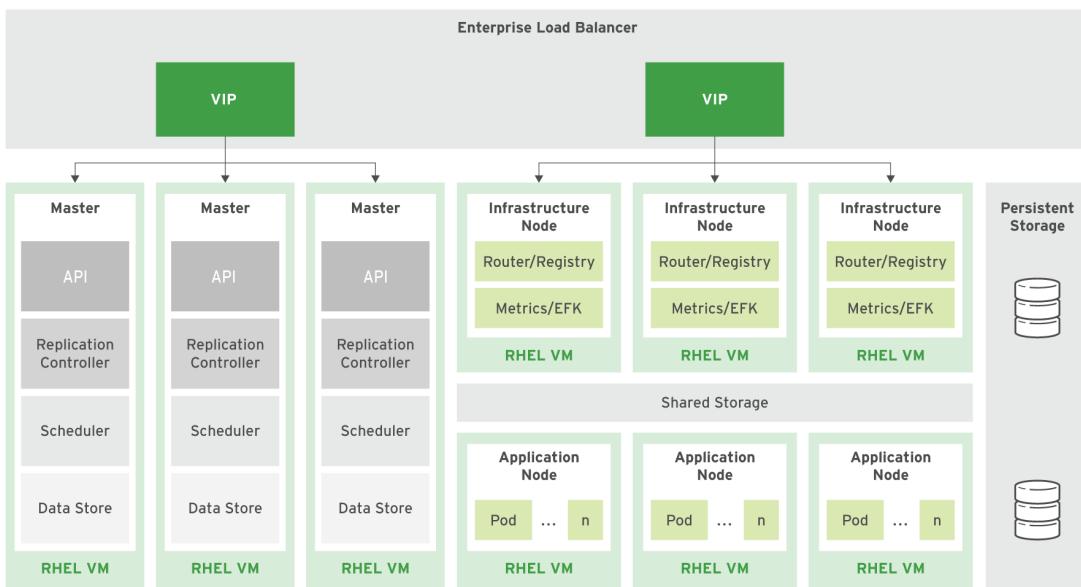


Figure 1.2: OpenShift Container Platform HA cluster architecture

The diagram *Figure 1.2: OpenShift Container Platform HA cluster architecture* describes the architecture of the environment used during this course. It consists of three master servers and six nodes. The nodes are subdivided into two groups. Three of node servers are designated to run the OpenShift router service, image registry, and other infrastructure services. These are referred to as infrastructure nodes. The remaining nodes run the actual application workloads and are referred to as application nodes or compute nodes. To designate which nodes do which job, labels are assigned to each node. OpenShift/Kubernetes uses labels to assign certain workload types, such as *infra* or *primary* workload types.

Non-HA Cluster versus HA Cluster

The difference between a non-HA cluster and one that is highly available is that the HA cluster has three or more master nodes that are accessed through the enterprise-level load balancer. Additionally, the enterprise level load balancer provides access to multiple infrastructure nodes that host router services to the pods and image registry services that use shared storage backed by persistent storage.

The following table describes the typical differences between a non-HA cluster and an HA cluster:

	OpenShift Cluster	OpenShift HA Cluster
Master servers	A single master server.	Three or more master servers.
Load balancing	A service load-balances client requests among member pods on the application nodes.	A single virtual IP address provides access for enterprise-level load balancing of master servers. A separate virtual IP address provides access for enterprise-level load balancing of the infrastructure nodes.

	OpenShift Cluster	OpenShift HA Cluster
Shared storage	Local storage or external persistent storage.	A single storage volume shared by all infrastructure nodes, and based on persistent storage.



References

Further information about OpenShift HA is available in the Architecture documentation for *OpenShift Container Platform* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Further information about OpenShift HA is available in the Cluster Administration documentation for *OpenShift Container Platform* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Further information about OpenShift HA is available in the Installation and Configuration documentation for *OpenShift Container Platform* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Quiz: Describing Highly Available OpenShift Architecture

Choose the correct answers to the following questions:

1. Which one of the following statements describe the primary reason for deploying a native HA method OpenShift cluster (Choose one)?
 - a. To ensure that, when the master service fails, the system can continue to respond to application failures or the creation of new applications.
 - b. To ensure that the etcd data store is fully redundant with an active-passive availability style.
 - c. To ensure that the cluster has no single point of failure.
 - d. To enhance productivity of a 2-node master server configuration.
2. Which event triggers the automatic configuration of the native HA method using HAProxy during the advanced installation method?
 - a. Setting the availability style of the master server to active-active.
 - b. Setting the availability style of the HAProxy server to active-active.
 - c. Requesting enterprise-level load balancing.
 - d. Requesting three or more master servers.
3. Which master component watches the etcd data store for changes to replication controller objects to modify the desired state?
 - a. Controller Management Server
 - b. API Server
 - c. HAProxy Server
 - d. Enterprise Load Balancer
4. Which master component in a highly available cluster assigns pods to nodes, synchronizes pod information, and is managed by HAProxy?
 - a. OpenShift Controller services
 - b. API service
 - c. HAProxy service
 - d. Enterprise Load Balancer

Solution

Choose the correct answers to the following questions:

1. Which one of the following statements describe the primary reason for deploying a native HA method OpenShift cluster (Choose one)?
 - a. To ensure that, when the master service fails, the system can continue to respond to application failures or the creation of new applications.
 - b. To ensure that the etcd data store is fully redundant with an active-passive availability style.
 - c. **To ensure that the cluster has no single point of failure.**
 - d. To enhance productivity of a 2-node master server configuration.
2. Which event triggers the automatic configuration of the native HA method using HAProxy during the advanced installation method?
 - a. Setting the availability style of the master server to active-active.
 - b. Setting the availability style of the HAProxy server to active-active.
 - c. Requesting enterprise-level load balancing.
 - d. **Requesting three or more master servers.**
3. Which master component watches the etcd data store for changes to replication controller objects to modify the desired state?
 - a. **Controller Management Server**
 - b. API Server
 - c. HAProxy Server
 - d. Enterprise Load Balancer
4. Which master component in a highly available cluster assigns pods to nodes, synchronizes pod information, and is managed by HAProxy?
 - a. OpenShift Controller services
 - b. **API service**
 - c. HAProxy service
 - d. Enterprise Load Balancer

Determining Networking and Sizing Requirements

Objective

After completing this section, students should be able to determine the networking and sizing requirements for an HA cluster.

Sizing Considerations and Calculations

It is important to determine how many nodes and pods you require for your OpenShift cluster. Too many pods may affect the node's performance and create bottlenecks. Cluster scalability correlates to the number of pods in a cluster environment. The number of pods directly influences your setup.



Note

The following storage requirements and calculation guidelines differ between minor releases of OpenShift Container Platform. Therefore, the values suggested might may be different in the future.

For more information, consult the *Scaling and Performance Guide* listed in the references.

The following table provides the maximum supported limits for nodes and pods:

Type	Maximum
Maximum nodes per cluster	2000
Maximum pods per cluster	120,000
Maximum pods per node	250
Maximum pods per core	10

Sizing Calculations

To determine how many pods are expected to fit per node use the following formula:

$$\text{Maximum Pods per Cluster} / \text{Expected Pods per Node} = \text{Total Number of Nodes}$$

In a hypothetical scenario, you want the OpenShift cluster to run 2,200 pods. Using 250 as the maximum number of pods on each node, the result of the calculation is as follows:

$$2200 / 250 = 8.8$$

Rounding up the value, nine nodes are needed to run all the expected pods.

Alternatively, if you have a particular number of nodes in mind, you can use the following formula to determine how many pods will fit on each of the nodes:

```
Maximum Pods per Cluster / Total Number of Nodes = Expected Pods per Node
```

In this situation, if you expect to have 2,200 pods per cluster and you have 20 nodes, you can deploy 110 pods per node:

```
2200 / 20 = 110
```

Sizing Impacts on Networking

The OpenShift cluster needs two different network CIDRs defined in order to assign pod and service IP addresses to its own components and the workloads running on it. These two values are the *pod network CIDR* and the *services network CIDR*.

Both IP address ranges are virtual ranges, visible only inside of the OpenShift software-defined network (SDN). These IP addresses are not layer-3 routable and therefore do not require allocation externally within routing or switching infrastructures. The only requirement for selecting these ranges is that they do not conflict with real address spaces that this cluster, or applications running in the platform, may need to communicate with. For example, ranges that back-end databases, external services, and others, might occupy.

The following configuration excerpts are part of the Ansible inventory file used by the OpenShift advanced installer.

Network CIDR for pods

This value determines the maximum number of pod IP addresses available to the cluster. The default value of /14 will provide 262,142 pod IP addresses for the cluster to assign to pods.

```
osm_cluster_network_cidr=10.128.0.0/14
```

Network CIDR for services

Each service in the cluster is assigned an IP address from this range. The default value of /16 provides up to 65,534 IP addresses for services.

```
openshift_portal_net=172.30.0.0/16
```

Defaults

The **osm_cluster_network_cidr** and **openshift_portal_net** variables default to /14 and /16, if not set. If you prefer to have your inventory file explicitly describe your environment, however, Red Hat recommends that you set them.

One method for calculating the total number of hosts that have an IP address on a specific subnet is to subtract the network mask from the 32-bit network address range, and then use the remainder as an exponent of base two.

```
Network address/CIDR: 10.100.0.0/14 (as defined by the osm_cluster_network_cidr option)
Network address size = 32 bits
Network address mask = 14 bits
32 - 14 = 18-bits subnet
2 ^ 18 = 262144 total host IP addresses (including both network and broadcast addresses)
```

Or, from the command line:

```
[student@workstation ~]$ echo "2 ^ (32 - 14)"|bc
262144
```

Master Service Ports

Master API port

This is the port that the master API service listens on.

```
openshift_master_api_port=8443
```

Master console port

This is the port that the master console service listens on.

```
openshift_master_console_port=8443
```

Defaults

The `openshift_master_api_port` and `openshift_master_console_port` variables default to port 8443. When using dedicated hosts for the masters, however, you can set these values to 443 and omit the port number from URLs when connecting to the service.

SDN Plug-ins

OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network. This enables communication between pods across the OpenShift Container Platform cluster. This pod network is established and maintained by the OpenShift SDN, which configures an overlay network using Open vSwitch (OVS).

OpenShift SDN provides three SDN plug-ins for configuring the pod network:

SDN Plug-in	Description
ovs-subnet	The original plug-in which provides a "flat" pod network where every pod can communicate with every other pod and service.
ovs-multitenant	Provides OpenShift Container Platform with project-level isolation for pods and services. Each project receives a unique Virtual Network ID (VNID) which identifies traffic from pods assigned to the project. Pods from different projects cannot send packets to or receive packets from pods and services of a different project.
ovs-networkpolicy	Allows project administrators to configure their own isolation policies using NetworkPolicy objects.

Router Subdomain and Other DNS Requirements

All hosts in the cluster need to be resolvable using DNS. Additionally, if using a control node as the Ansible installer, it too must be able to resolve all hosts in the cluster.

DNS Requirements

Typically, in an HA cluster, there are two DNS names for the load-balanced IP address that points to the three master servers for access to the API, CLI, and console services. One of these names is the public name that users use to log in to the cluster. The other is an internal name that is used by internal components within the cluster to communicate with the master. These values must also resolve, and be placed in the Ansible inventory file used by the OpenShift advanced installer.

Public master host name

This is the host name that external users and tools or both use to log in to the OpenShift cluster.

```
openshift_master_cluster_public_hostname=console.mycluster.example.com
```

Internal master host name

This is the host name that users and tools or both use to log in to the OpenShift API and web console.

```
openshift_master_cluster_hostname=console.mycluster.example.com
```

Wildcard DNS entry for infrastructure (router) nodes

In addition to the host names for the master console and API, a wildcard DNS entry must exist under a unique subdomain, such as, ***.cloudapps.mycluster.example.com**, that resolves to the IP addresses of the three infrastructure nodes via **A** records, or to their host names via **CNAME** records. The subdomain allows new routes to be automatically routable to the cluster under the subdomain, such as **mynewapp.cloudapps.mycluster.example.com**.

```
openshift_master_default_subdomain=cloudapps.mycluster.example.com
```

If a wildcard DNS entry does not exist, every exposed route would require the entry to be created in order to route it to the OpenShift cluster. If this is desired, Red Hat strongly recommends that you implement automated integration between OpenShift and an external DNS system to automatically provision new DNS entries whenever a new route is created.

The following diagram describes how the load balancers route traffic to the master and infrastructure servers:

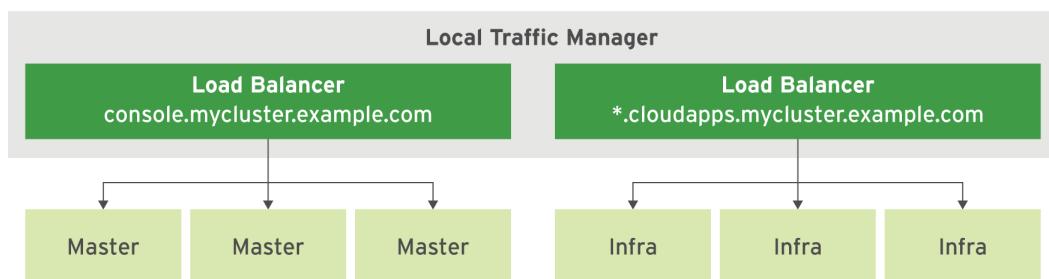


Figure 1.3: Load balancer traffic management



References

Further information about OpenShift HA is available in the Installation and Configuration documentation for *OpenShift Container Platform* at

| https://access.redhat.com/documentation/en-us/openshift_container_platform/

Further information about sizing guidelines is available in the *Scaling and Performance Guide for OpenShift Container Platform* at

| https://access.redhat.com/documentation/en-us/openshift_container_platform/

Guided Exercise: Determining Networking and Sizing Requirements

In this exercise, you will determine and document the network and node requirements for installing an OpenShift cluster.

Outcomes

You should be able to calculate the network and node requirements for an OpenShift cluster.

Before you begin

The following diagram describes a hypothetical OpenShift cluster environment. Use this diagram as a guideline as you determine the values required for this exercise.

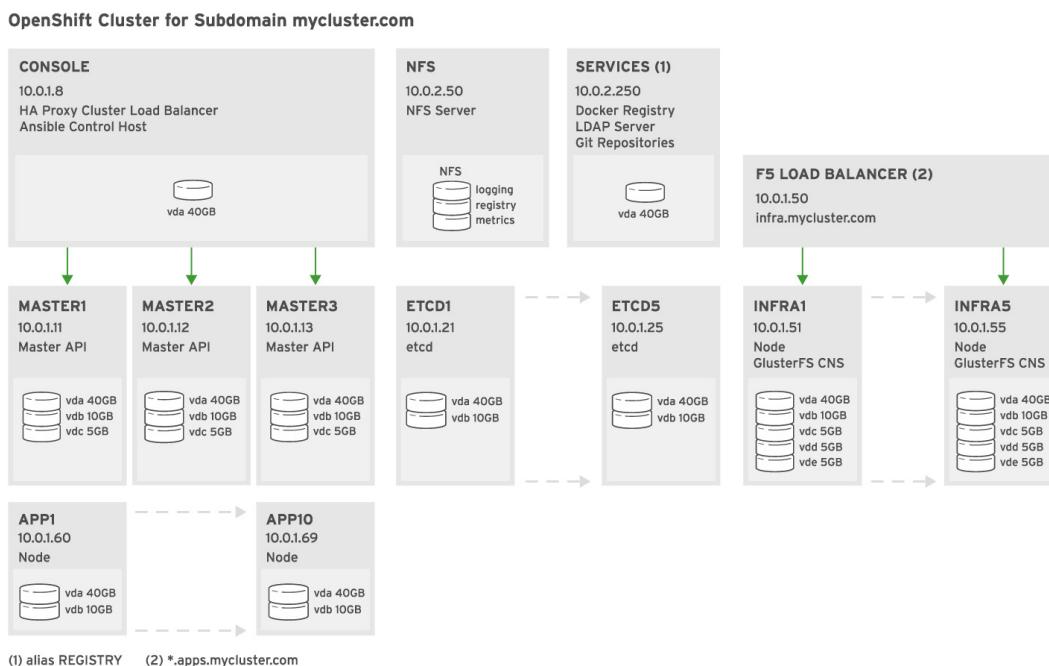


Figure 1.4: OpenShift cluster hypothetical design

Steps

- Determine the total number of nodes required to run 10,000 pods on a cluster that is capable of running 250 pods on each node.
 - Given the number of total pods per cluster and expected number of pods per node, you can determine how many nodes are needed using the following formula:

$$\text{Maximum Pods per Cluster} / \text{Expected Pods per Node} = \text{Total Number of Nodes}$$

- Write the results for the total number of nodes in the space provided:

Documentation

Total number of nodes	10 000/250 = 40
-----------------------	------------------------

For the amount of pods running, the supported amount of nodes is 40 nodes.

2. Determine the network CIDRs that will support the total number of nodes and allow the cluster to scale.
 - 2.1. This value is determined by calculating the total number host IP addresses available when using a specific netmask. The goal is to use a netmask that allows the cluster to scale the number of pods as needed. For example, using a 18-bit netmask allows for 16,382 host IP addresses for pods. This leaves enough room to scale but may be more than necessary.
 - 2.2. Calculate the best network CIDR for your cluster and write your results in the space provided:

Documentation	
Network address in CIDR notation	(2 ^ (32 - 18)) - 2 = 16382

3. Determine and document the service ports that are used by the API and console node.
 - 3.1. Refer to the lecture content as a guideline to document the service ports for your environment. Write your findings in the spaces provided:

Documentation	
API Service port	8443
Console node service port	8443

4. Determine the subdomain and host names for your environment.
 - 4.1. Write the subdomain, the host name for the public master, and the wildcard DNS entry for the infrastructure nodes in the spaces provided:

Documentation	
Subdomain	apps.mycluster.com
Public master host name	console.lab.example.com
Wildcard DNS entry for infrastructure nodes	*.apps.mycluster.com

This concludes the guided exercise.

Determining Load Balancing and Pod Distribution Schemes

Objective

After completing this section, students should be able to determine the load balancing and pod distribution schemes for an HA cluster.

Load Balancing Master Nodes

In OpenShift, to ensure that the environment is running in a highly available (HA) cluster, all master nodes must be accessible through a single IP address, using a load balancer. This approach guarantees a uniform access to the REST API and allows a unified view to the web console.

Use a load balancer to provide a single address and to balance requests across all master nodes.

The OpenShift advanced installation method can be customized to use HAProxy as the default load balancer. Even though HAProxy is a reliable tool and supports complex deployment scenarios, the HAProxy configuration files may become complex.

OpenShift also supports any hardware-based load balancer. This has some advantages, such as:

- Easier configuration for complex architectures
- Built-in and proven capabilities to fail over a single virtual IP (VIP) address

Load Balancing Infrastructure Nodes

Nodes running essential services for OpenShift must be load balanced to support high availability. For example, if the OpenShift router fails, the load balancer can redirect requests to another router running on a different node.

In a similar approach to the master nodes, a load balancer must be configured to minimize the downtime of any of the services running on the infrastructure nodes. You can configure OpenShift to use either HAProxy or a hardware load balancer to manage requests to all infrastructure nodes.

Pod Distribution Schemes

To enforce better pod management, OpenShift separates pods used internally by OpenShift, such as the registry and the registry console and the remaining pods. This strategy allows:

- *Improved pod performance:* Service pods request more CPU time to support the deployment of application pods on multiple nodes. To minimize the impact on those application pods, the service pods are scheduled to run on a dedicated node.
- *Simplified maintenance:* Service pods may need to run some maintenance tasks, such as container image or registry cleanup. Stopping a node that is running both application and service pods can cause an application to become unavailable.

OpenShift enforces this organization by creating node labels that differentiate an infrastructure node (by labeling it as **region=infra**) from an application node (by labeling it as

region=apps). Further customization can be done to segregate even more application pods. For example, high-demand applications can be deployed on nodes labeled **high-cpu-apps**, which have higher processing power than those labeled **low-cpu-apps**.



References

Further information is available in the Routing from Edge Load Balancers chapter of the *OpenShift Installation and Configuration Guide* for OpenShift Container Platform 3.6 at

|| <https://access.redhat.com/documentation/en-US/index.html>

Further information about the Ansible variables related to node selectors is available in the Advanced Installation chapter of the *OpenShift Container Platform Cluster Installation and Configuration Guide* at

|| <https://access.redhat.com/documentation/en-US/index.html>

Quiz: Determining Load Balancing and Pod Distribution Schemes

Choose the correct answers to the following questions:

1. You are going to deploy OpenShift in an environment running 20 nodes. Five nodes will run master nodes and the remaining ones will run as non-master-nodes. Which two statements are true? (Choose two.)
 - a. The OpenShift installer configures all 20 nodes to support application pod deployments.
 - b. OpenShift automatically configures HAProxy to run as the default load balancer.
 - c. OpenShift can deploy pods on any of the 14 non-master nodes. One node is used as the HAProxy server.
 - d. Since OpenShift Container Platform cannot differentiate master nodes from application nodes, application nodes must be installed after the master nodes are deployed.
2. Which two statements describe characteristics of OpenShift Container Platform? (Choose two.)
 - a. OpenShift Container Platform can segregate applications from infrastructure nodes only by defining the application node as schedulable.
 - b. OpenShift Container Platform application nodes can be configured by defining them with the **apps** label.
 - c. OpenShift Container Platform infrastructure pods can be configured by defining them as unscheduled nodes.
 - d. OpenShift Container Platform infrastructure pods can be configured by defining them with the **infra** label.
3. Which statement describes characteristics of OpenShift Container Platform?
 - a. OpenShift Container Platform only supports software-based load balancers.
 - b. OpenShift Container Platform only supports hardware-based load balancers.
 - c. OpenShift Container Platform supports every hardware- and software-based load balancer available on the market.
 - d. OpenShift Container Platform supports HAProxy as the default software-based load balancer.

Solution

Choose the correct answers to the following questions:

1. You are going to deploy OpenShift in an environment running 20 nodes. Five nodes will run master nodes and the remaining ones will run as non-master-nodes. Which two statements are true? (Choose two.)
 - a. The OpenShift installer configures all 20 nodes to support application pod deployments.
 - b. **OpenShift automatically configures HAProxy to run as the default load balancer.**
 - c. **OpenShift can deploy pods on any of the 14 non-master nodes. One node is used as the HAProxy server.**
 - d. Since OpenShift Container Platform cannot differentiate master nodes from application nodes, application nodes must be installed after the master nodes are deployed.
2. Which two statements describe characteristics of OpenShift Container Platform? (Choose two.)
 - a. OpenShift Container Platform can segregate applications from infrastructure nodes only by defining the application node as schedulable.
 - b. **OpenShift Container Platform application nodes can be configured by defining them with the `apps` label.**
 - c. OpenShift Container Platform infrastructure pods can be configured by defining them as unscheduled nodes.
 - d. **OpenShift Container Platform infrastructure pods can be configured by defining them with the `infra` label.**
3. Which statement describes characteristics of OpenShift Container Platform?
 - a. OpenShift Container Platform only supports software-based load balancers.
 - b. OpenShift Container Platform only supports hardware-based load balancers.
 - c. OpenShift Container Platform supports every hardware- and software-based load balancer available on the market.
 - d. **OpenShift Container Platform supports HAProxy as the default software-based load balancer.**

Lab: Designing a Highly Available Cluster

In this lab, you will determine and document the requirements for deploying an OpenShift cluster.

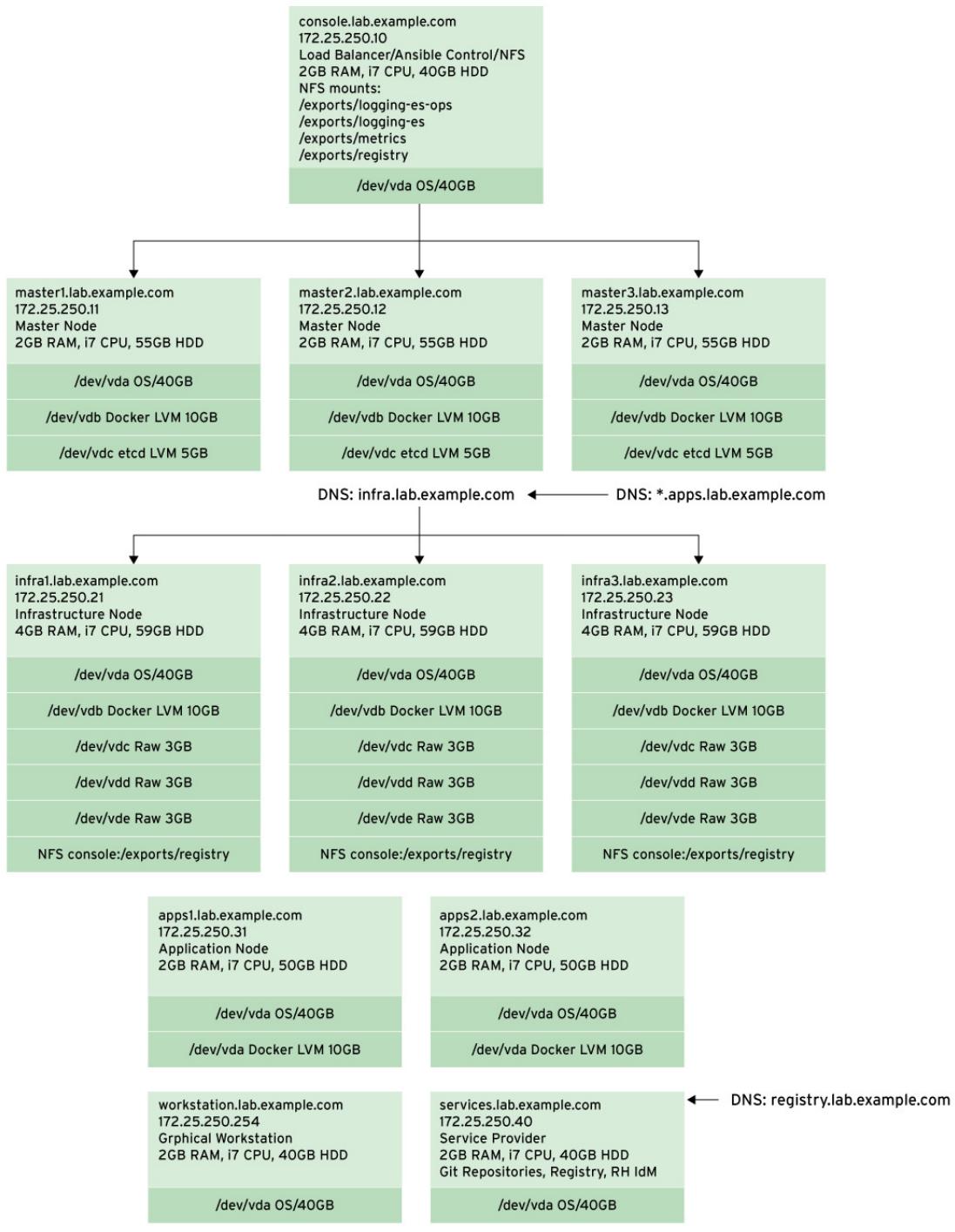
Outcomes

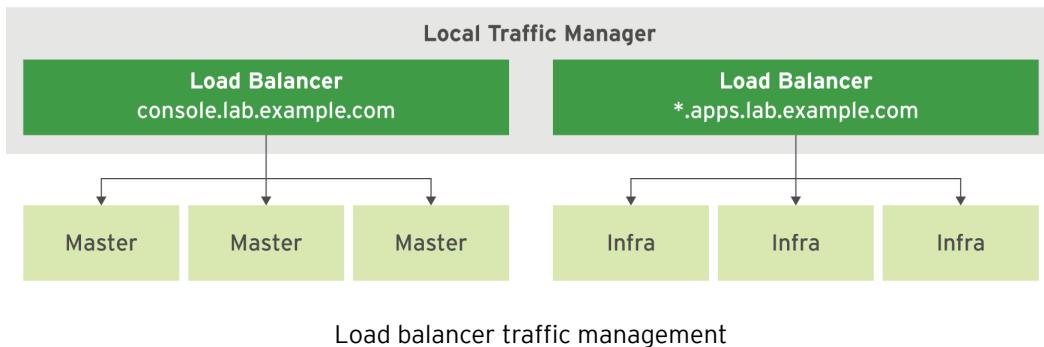
You should be able to:

- Determine sizing for all nodes in an OpenShift cluster.
- Calculate the number of pods expected per node in an OpenShift cluster.
- Determine the network CIDR for pods and services.
- Assign port numbers to master services.
- Determine the DNS requirements for an OpenShift cluster.

Before you begin

The following diagrams describe a hypothetical OpenShift cluster environment. Use these diagrams as a guideline as you determine the values required for this lab:



**Steps**

1. You have three application nodes. Determine the total number of pods that can run on each node when you have an expectation of running 750 pods on a cluster.

Documentation	
Total number of nodes	3
Total number of pods per node	

2. Determine the network CIDR for pods and services that supports the total number of nodes and allows the cluster to scale.

Documentation	
Pod network address in CIDR notation	
Service network address in CIDR notation	

3. Determine the subdomain and host names for the master and infrastructure nodes.

Documentation	
Subdomain	
Public master host name	
Wildcard DNS entry for infrastructure nodes	

This concludes the lab.

Solution

In this lab, you will determine and document the requirements for deploying an OpenShift cluster.

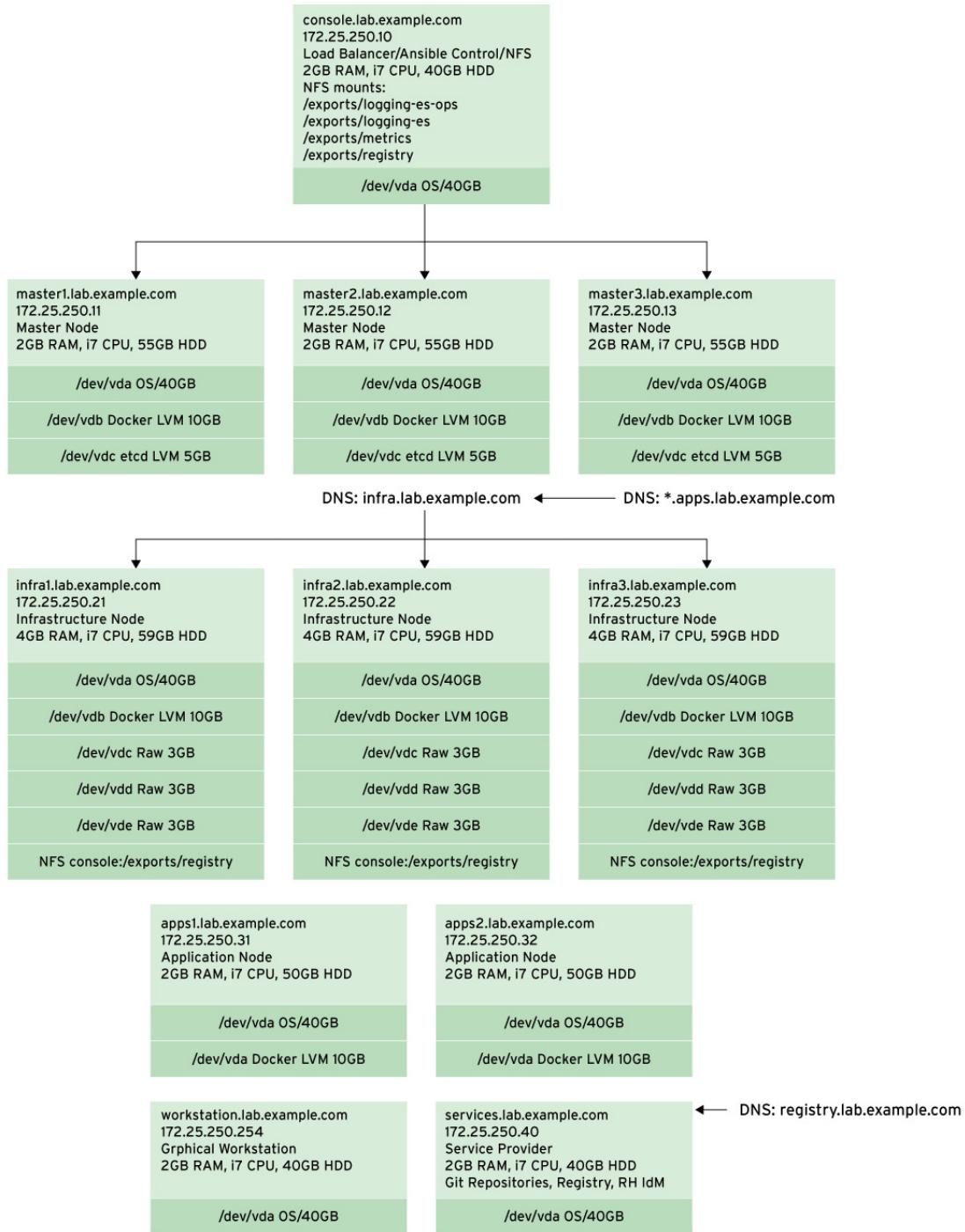
Outcomes

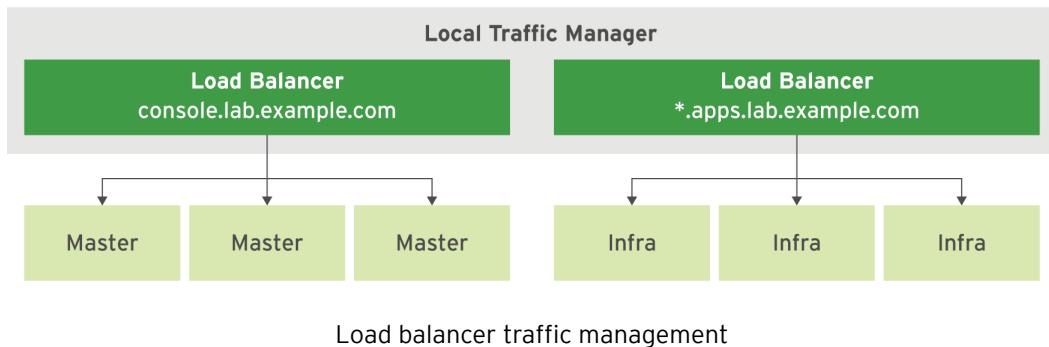
You should be able to:

- Determine sizing for all nodes in an OpenShift cluster.
- Calculate the number of pods expected per node in an OpenShift cluster.
- Determine the network CIDR for pods and services.
- Assign port numbers to master services.
- Determine the DNS requirements for an OpenShift cluster.

Before you begin

The following diagrams describe a hypothetical OpenShift cluster environment. Use these diagrams as a guideline as you determine the values required for this lab:





Load balancer traffic management

Steps

1. You have three application nodes. Determine the total number of pods that can run on each node when you have an expectation of running 750 pods on a cluster.

Documentation	
Total number of nodes	3
Total number of pods per node	250

- 1.1. Use the following formula to determine the expected pods per node:

$$\text{Maximum Pods per Cluster} / \text{Total Number of Nodes} = \text{Expected Pods per Node}$$

The calculation to determine the total number of pods per node is as follows:

$$750 / 3 = 250$$

2. Determine the network CIDR for pods and services that supports the total number of nodes and allows the cluster to scale.

Documentation	
Pod network address in CIDR notation	10.100.0.0/20
Service network address in CIDR notation	172.30.0.0/20

- 2.1. The value is determined by calculating the total number of host IP addresses available when using a specific netmask. The goal is to use a netmask that allows the cluster to scale the number of pods as needed.

One method for calculating the total number of hosts within one subnet is to subtract the network mask from the 32-bit network range, and then use the remainder as an exponent of base two.

For example:

```
10.100.0.0/20 (as defined by the osm_cluster_network_cidr option )
32 - 20 = 12
(2 ^ 12) - 2 = 4094 (excluding the network and broadcast addresses)
```

Allowing for 4094 IP addresses for pods and services.

3. Determine the subdomain and host names for the master and infrastructure nodes.

Documentation	
Subdomain	lab.example.com
Public master host name	console.lab.example.com
Wildcard DNS entry for infrastructure nodes	*.apps.lab.example.com

- 3.1. Set the subdomain, master, and infrastructure host names to match the classroom environment.

- The subdomain for all student workstations in the classroom is **lab.example.com**
- The host name for the master API and web console server is
console.lab.example.com
- The wildcard DNS entry for infrastructure nodes is ***.apps.lab.example.com**

This concludes the lab.

Summary

In this chapter, you learned:

- An OpenShift cluster consists of a master server that manages nodes within the cluster and node servers that run schedules pods. The node servers are application nodes that provide runtime environments for containers.
- The primary objective when configuring a highly available cluster is to ensure that there is no single point of failure.
- Correctly sizing the number of nodes and pods that run in the cluster ensures optimal node performance and prevention of bottlenecks.
- OpenShift Container Platform uses a software-defined networking (SDN) approach to provide a unified cluster network. This enables communication between pods across the OpenShift cluster.
- The OpenShift cluster has two different network CIDRs defined that assign pod and service IP addresses to its own components as well as the workloads running on it. These two values are the pod network CIDR and the services network CIDR. Both network address ranges are virtual ranges, visible only inside of the OpenShift software-defined network.
- In OpenShift, to ensure that the environment is running on a highly available (HA) cluster, a load balancer is used to provide a single address for all master nodes and to balance requests to available masters. This approach guarantees uniform access to the REST API and allows a unified view to the web console.



CHAPTER 2

PREPARING TO INSTALL AN HA CLUSTER

Overview	
Goal	Configure the advanced installer and prepare the cluster environment for HA installation.
Objectives	<ul style="list-style-type: none">Describe the functionality and execution of the advanced installation method.Build an Ansible inventory file for the advanced installation method.Configure the Ansible inventory file to request the installation of a persistent integrated registry.Configure the Ansible inventory and cluster environment to run disconnected from the Internet.
Sections	<ul style="list-style-type: none">Describing the Advanced Installation Method (and Quiz)Building the Ansible Inventory (and Guided Exercise)Configuring the Integrated Registry (and Guided Exercise)Preparing a Disconnected Environment (and Guided Exercise)
Lab	Preparing to Install an HA Cluster

Describing the Advanced Installation Method

Objective

After completing this section, students should be able to describe the functionality and execution of the advanced installation method.

OpenShift Ansible Playbooks

OpenShift Container Platform provides two different installation methods, based on Ansible, for deploying an OpenShift Platform Container cluster; the *quick* method and the *advanced* method. Both methods are supported for production, which means that you can chose the method that suits your deployment goals. If administrators want to create a proof of concept deployment with OpenShift Container Platform, the quick installation method is recommended. The quick installation method provides an interactive command-line interface that prompts for configuration options applicable to the environment. The advanced installation method allows for more control over the cluster's configuration.



Important

Ansible is not available for RHEL Atomic Host. It must be installed on RHEL 7.

The advanced installation method provides two types of playbooks; bring your own (BYO), and playbooks for cloud providers. Both types are located in the **/usr/share/ansible/openshift-ansible/playbooks/** directory. Bring your own (BYO) playbooks allow for customization, and cloud provider playbooks provide the files required for configurations to specific cloud providers. Each playbook type has a separate and distinct playbook directory structure.

```
[user@demo playbooks]$ tree -d byo
byo
├── openshift-cfme①
├── openshift-checks②
│   └── certificate_expiry
│       └── roles -> ../../../../roles
├── openshift-cluster③
│   ├── roles -> ../../..roles
│   └── upgrades
│       ├── docker
│       ├── v3_3
│       ├── v3_4
│       ├── v3_5
│       └── v3_6
└── openshift-etcd④
    └── roles -> ../../..roles
├── openshift-glusterfs⑤
    └── roles -> ../../..roles
├── openshift-master
    └── roles -> ../../..roles
├── openshift-node
    └── roles -> ../../..roles
└── openshift-preflight
```

```
└─ roles -> ../../roles
```

- ① Configures a CloudForms appliance in OpenShift.
- ② Detects problems prior to an installation and performs health checks on existing OpenShift clusters.
- ③ Configures roles for an OpenShift cluster.
- ④ Configures an Etcd server or cluster to store configuration and systems data.
- ⑤ Enables pods to use GlusterFS volumes in OpenShift.

```
[user@demo playbooks]$ tree -d aws
aws
└─ openshift-cluster①
    ├─ library
    ├─ roles -> ../../../../roles
    ├─ tasks
    └─ templates
```

- ① Deploys OpenShift clusters for AWS with auto-scale groups and custom Amazon Machine Images (AMIs).

To run either type of playbook, use the following command:

```
[user@demo playbooks]$ ansible-playbook -i inventory-file playbook-file
```

Inventory File

The advanced installation method deploys multiple servers simultaneously by using the inventory file for the playbook used to install OpenShift Container Platform. The inventory file contains a list of the systems Ansible interacts with during the deployment. Depending on the requirements of the deployment, the inventory file could include host names or implement host and group variables for more complex deployments such as an OpenShift Container Platform cluster. The default inventory file is stored in **/etc/ansible/hosts** and uses an INI format by default. An alternate inventory file can also be specified by using the **-i** option.

Cluster Variables

The host and cluster variables in the inventory file drive the installation process. Environment variables are assigned on each host during the advanced installation method through the default Ansible inventory file. These variables are defined in the **[masters]** or **[nodes]** section. For example, to specify the system **master1.mycluster.com** as a *master* within the cluster add the following entry.

```
[masters]
master1.mycluster.com
```

The cluster variables used during the advanced installation method are used globally and apply to the OpenShift Container Platform deployment as a whole. These variables are specified in the cluster hosts file on separate, single lines in the **[OSEv3:vars]** section. For example:

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise①
openshift_release=v3.6②
openshift_image_tag=v3.6.173.0.49③
```

- ① Specifies the deployment type. Valid values are **origin** and **openshift-enterprise**.
- ② Specifies the generic release of OpenShift to install. It is suited for containerized installations where it is used to look up the latest version of the container images. That version number is the same tag that is used to configure the cluster.
- ③ Specifies the exact container image tag to install or configure.



Warning

The **openshift_image_tag** value is used for all hosts in containerized environments, even those that have another version installed. This could potentially trigger an upgrade and subsequent downtime, so be careful modifying this value after the cluster has been configured.

The previous example lists a small subset of cluster variables. There are many more. A complete list of cluster-wide variables can be found at <https://github.com/openshift/openshift-ansible/blob/master/inventory/hosts.example>.

Cluster Node Components

In OpenShift Container Platform all hosts are considered OpenShift nodes, including the master nodes. All hosts, within the design of Kubernetes, have some basic pieces in common which make them part of the node component. These basic pieces are required to run everywhere. An OpenShift cluster consists of the following components, all of which should be running on all master and node hosts:

- *Docker*: All OpenShift packages require the **docker** service as a dependency and the **systemd** service should be set to **active** and **enabled** on all the hosts.
- *OpenVSwitch (OVS)*: Every host requires Open VSwitch to support OpenShift SDN communications. The **systemd** service should be set to **active** and **disabled**.



Note

The Open VSwitch service is disabled in *systemd* because it is a dependency of the OpenShift Node service, and should **NOT** be started or stopped independently of that service.

- *OpenShift node*: The **atomic-openshift-node** service should be set to **active** and **enabled**.

Single-master Cluster

In a single-master cluster, all the cluster components are grouped together in a single package and the **atomic-openshift-master** service should be set to **active** and **enabled** on all hosts.

Multi-master (Native High Availability)

In a multi-master cluster deployment that uses native high availability, the master components are separated. This is because they have varied requirements when there is more than one instance of them.

- **Etcd:** This component is a data store and is responsible for holding the state data of the cluster. Due to the requirement of at least three active cluster members in an HA cluster, **Etcd** has its own service. The service should be set to **active** and **enabled**.
- **Master API:** This component is responsible for handling requests from clients, including nodes, users, administrators, and other infrastructure systems deployed to OpenShift. The master API is a stateless component, and therefore the service should be set to **active** and **enabled** on all masters.
- **Master controllers** - The master controller consists of two components, the OpenShift scheduler and the replication controller. These components run in either an active or passive mode, and are responsible for the placement and maintenance of pods in the cluster. This mitigates the chances that more than one controller might attempt to re-create the failed pod. This may result in the **atomic-openshift-master-controllers** service displaying as inactive or failed at particular instances in the cluster life cycle. At least one controller must always be active, and all controllers should have the **atomic-openshift-master-controllers** service enabled.

Ansible Ad Hoc Commands

Ansible uses modules to perform the configuration of the nodes defined in the inventory. Prior to configuring these nodes, Ansible can use modules such as the **ping** module which tests login, and for a usable version of python prior to configuration these nodes. You can also use Ansible ad hoc commands to configure partitions, volumes, file systems, and perform other pre-installation tasks. The following command connects to all hosts listed in the **hosts-demo** file.

```
[user@demo playbooks]$ ansible -i hosts-demo OSEv3 -m ping
```

To install and configure Docker service to use the **/dev/vdc** volume for all local docker storage run the following command.

```
[user@demo playbooks]$ ansible -i hosts-demo nodes -m shell \
-a 'echo "DEVS=/dev/vdc" > /etc/sysconfig/docker-storage-setup'
```

Pre and Post Configurations

Red Hat recommends creating custom Ansible playbooks for pre and post configurations instead of using shell scripting or manual commands. For example, instead of constructing a bash script to configure the yum repositories on all the nodes in the cluster, we can create an Ansible playbook to configure the yum repositories.

```
- hosts: all
become: yes
tasks:
- name: Define Yum repositories
  yum_repository:
    name: "{{item.name}}"
    gpgcheck: "{{item.gpgcheck}}"
    description: "{{item.desc}}"
    baseurl: "{{item.baseurl}}"
  with_items:
    - name: 'rhel-7-server-extras-rpms'
      desc: 'Remote copy of RHEL Extras RPMS'
      gpgcheck: false
      baseurl: 'http://demo.example.com/ocp3.6/x86_64/extras'
    - name: 'rhel-7-server-ose-3.6-rpms'
```

```
gpgcheck: false
desc: 'Remote copy of OCP RPMS'
baseurl: 'http://demo.example.com/ocp3.6/x86_64/ocp'
- name: 'rhel-7-fast-datapath-rpms'
  gpgcheck: false
  desc: 'Remote copy of Fast Datapath RPMS'
  baseurl: 'http://demo.example.com/ocp3.6/x86_64/fast'
- name: 'ocp-7-server-ose-3.6-rpms'
  gpgcheck: false
  desc: 'Remote copy of OCP extra RPMS'
  baseurl: 'http://demo.example.com/ocp3.6/x86_64/ocp-extras'
```

References

Further information is available in the Installing a cluster chapter of the *OpenShift Container Platform 3.6* for Red Hat Enterprise Linux 7 at
<https://access.redhat.com/documentation/en-US/index.html>

Quiz: Describing the Advanced Installation Method

Choose the correct answers to the following questions:

1. What are the two installation methods provided by OpenShift Container Platform? (Choose one)
 - a. x86_64 and i386
 - b. Container and remote
 - c. Quick and advanced
 - d. Minimal and full

2. What is contained within the **openshift-cfme** directory? (Choose one.)
 - a. Ansible Playbooks responsible for detecting problems.
 - b. Ansible Playbooks responsible for the configuration of an OpenShift cluster.
 - c. Ansible Playbooks responsible for enabling pods to use GlusterFS volumes in OpenShift.
 - d. Ansible Playbooks responsible for configuring CloudForms Management Engine (CFME) in OpenShift.

3. What type of playbooks are provided by the advanced installation method? (Choose two.)
 - a. AWS
 - b. OpenStack
 - c. Bring your own (BYO)
 - d. OpenShift
 - e. Atomic
 - f. Cloud provider

4. Which cluster variable specifies the exact container image tag to install or configure? (Choose one.)
 - a. **openshift_release**
 - b. **openshift_image_tag**
 - c. **openshift_image_version**
 - d. **None**

5. Which component is a data store that is responsible for holding the state data of the cluster? (Choose one.)
 - a. Master API
 - b. Etcd
 - c. Master controller
 - d. Docker

Solution

Choose the correct answers to the following questions:

1. What are the two installation methods provided by OpenShift Container Platform? (Choose one)
 - a. x86_64 and i386
 - b. Container and remote
 - c. **Quick and advanced**
 - d. Minimal and full
2. What is contained within the **openshift-cfme** directory? (Choose one.)
 - a. Ansible Playbooks responsible for detecting problems.
 - b. Ansible Playbooks responsible for the configuration of an OpenShift cluster.
 - c. Ansible Playbooks responsible for enabling pods to use GlusterFS volumes in OpenShift.
 - d. **Ansible Playbooks responsible for configuring CloudForms Management Engine (CFME) in OpenShift.**
3. What type of playbooks are provided by the advanced installation method? (Choose two.)
 - a. AWS
 - b. OpenStack
 - c. **Bring your own (BYO)**
 - d. OpenShift
 - e. Atomic
 - f. **Cloud provider**
4. Which cluster variable specifies the exact container image tag to install or configure? (Choose one.)
 - a. `openshift_release`
 - b. **openshift_image_tag**
 - c. `openshift_image_version`
 - d. None
5. Which component is a data store that is responsible for holding the state data of the cluster? (Choose one.)
 - a. Master API
 - b. **Etcd**
 - c. Master controller
 - d. Docker

Building the Ansible Inventory

Objective

After completing this section, students should be able to build an Ansible inventory file for the advanced installation method.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Ansible Inventory

The OpenShift Container Platform advanced installation uses playbooks that require host names and variables to be defined in an inventory file, which supports the installation. The inventory file contains the configuration for the OpenShift Container Platform cluster. The default inventory file is **/etc/ansible/hosts** which must be either modified or created to reflect the desired OpenShift Container Platform infrastructure.

Building an Ansible Inventory

The initial building of an Ansible inventory begins with the creation of a project skeleton. The project skeleton is a directory used as a central point to manage the project. It contains the cluster subdirectory and inventory file. It is considered best practice for the cluster directory name to reflect the base domain of the cluster.

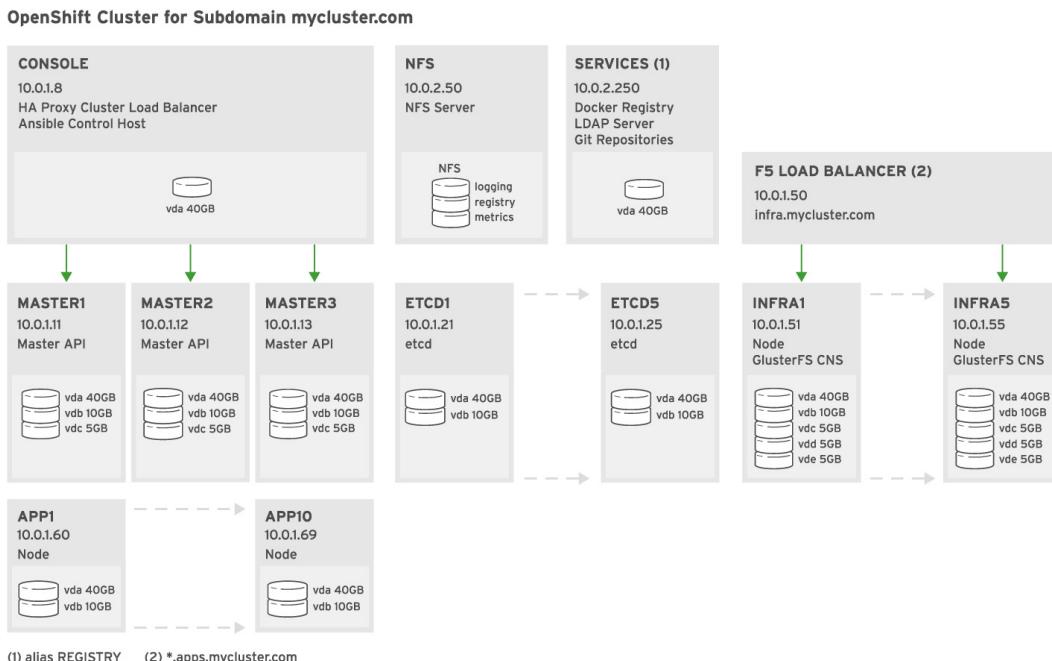


Figure 2.1: Theoretical cluster

For example, based on the OpenShift cluster diagram, the project directory is named **mycluster-openshift** and the cluster subdirectory is **mycluster.com**.

```
[root@demo ~]# mkdir /root/mycluster-openshift/mycluster.com
```

To guarantee a unique configuration source, Red Hat recommends that you add any configuration files to a source code repository, such as Git.

A single inventory hosts file is created in **/root/mycluster-openshift/mycluster.com**. The file name **hosts** is the standard name for the Ansible inventory file. Initially this file is configured with the base structure of the inventory file. The inventory file begins with the **[OSEv3:children]** group, which contains a list of installed host types, such as **masters**, **nodes**, **etcd**, **lb** (load balancer), and **nfs**. These host types translate to Ansible child groups, which can customize each host's configuration:

```
# hosts file for mycluster.com
[OSEv3:children]
masters
etcd
nodes
nfs
lb
```

Configuring the OpenShift Installation Version

To ensure all of the latest bug fixes and patches for a specific version are installed, it is recommended practice to decide on the major version of OpenShift to target, and to allow the installation playbook to take the latest minor release of that major version. To specify the OpenShift Container Platform deployment type and version to install, use the **openshift_deployment_type** and **openshift_release** variables respectively within the **[OSEv3:vars]** section.

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise①
openshift_release=v3.6②
```

- ① Defines the deployment type of the OpenShift product
- ② Defines the major version to be installed

OpenShift Container Platform Network Requirements

An OpenShift Container Platform cluster requires two different network CIDR values: pod network CIDR and services network CIDR. Both network CIDR values should be visible only inside the OpenShift software-defined network (SDN) and not be routable. These network CIDR values cannot conflict with the actual network ranges that are used by the cluster and its applications for communication.

Pod Network CIDR

The pod network CIDR value is defined by the **osm_cluster_network_cidr** variable. This variable overrides the SDN cluster network CIDR block, and pod IPs are assigned from this network. This network block must be a private block and must not conflict with existing network blocks in the cluster infrastructure. The default network is 10.128.0.0/14 and cannot be arbitrarily reconfigured after deployment, unless specific changes to it are made in the SDN master

configuration. If the variable is not configured then the default value is used, however Red Hat recommends that you set them explicitly in the inventory hosts file.

The pod network CIDR value is used to ascertain the number of pod IPs accessible to the cluster infrastructure. The default CIDR value is /14, which provides 262,142 pod IP addresses.

```
osm_cluster_network_cidr=10.128.0.0/14
```

Service Network CIDR

The service network CIDR value defines the network block used to provide IP addresses assigned to each service. This network block must be private and must not conflict with any existing network blocks in the infrastructure. The default network is 172.30.0.0/16, and cannot be reconfigured after deployment. If you need to change from the default value, avoid using 172.17.0.0/16. This is the network range that the **docker0** network adapter uses by default.

The service network CIDR range determines the IP addresses that are allocated to each service. The default CIDR value is /16 which provides up to 65,534 IP addresses.

```
openshift_portal_net=172.30.0.0/16
```

Master Service Port

The master API listens on the master service port defined by the variable **openshift_master_api_port**. Although the default is 8443, when using dedicated hosts as masters you can use port 443 and omit the port number from connecting URLs. The master console port is set by the **openshift_master_console_port** variable; the default port is 8443. The master console can also be set to use port 443, and the port number can be omitted from connecting URLs.

```
openshift_master_api_port=8443  
openshift_master_console_port=8443
```

Domain Name Service

An OpenShift high-availability cluster requires two DNS names; a public name and an internal name. The public and internal master host names are set by the **openshift_master_cluster_public_hostname** and **openshift_master_cluster_hostname** variables, respectively. The public name is used by users to log in to the cluster. The internal name is used by components within the cluster to reply to the master. Both the internal and public DNS names are for the load-balanced IP address, which points to the three master servers to access the console, CLI, and API services.

```
openshift_master_cluster_public_hostname=console.mycluster.com  
openshift_master_cluster_hostname=console.mycluster.com
```

Wildcard DNS

A wildcard DNS entry for infrastructure nodes enables any newly created routes to be automatically routable to the cluster under the subdomain. The wildcard DNS entry must exist in a unique subdomain, such as **apps.mycluster.com**, and resolve to either the host name or IP address of the infrastructure nodes. The inventory file variable that exposes the wildcard DNS entry is **openshift_master_default_subdomain**.

```
openshift_master_default_subdomain=apps.mycluster.com
```

Load Balancers

Adding an **1b** child group to the Ansible inventory file automatically configures HAProxy as the load balancer. To set a different host, use the **openshift_master_cluster_hostname** inventory file variable. To configure either the HAProxy load balancer or an external load balancer, customize the **openshift_master_cluster_method** variable as follows:

```
openshift_master_cluster_method=native  
openshift_master_cluster_hostname=console.mycluster.com
```

Node Labels

To configure the node labels, use the **openshift_node_labels** variable in the **[nodes]** child group in the Ansible inventory file, as follows:

```
# host group for nodes, includes region info  
[nodes]  
node.example.com openshift_node_labels="{'region': 'primary', 'zone': 'east'}"
```

Authentication

Master nodes in an OpenShift Container Platform infrastructure contain a built-in OAuth server for authentication. OAuth access tokens are requested by users to authenticate to the API. OAuth is configured using the master configuration file to specify an identity provider. The configured identity provider is used by the OAuth server to identify the user initiating the request. OAuth maps the user to the identity and creates an access token for the user.

Identity Providers

Identity providers are used for authentication and are configured using the master configuration file. The identity provider is configured in the Ansible inventory file using the **openshift_master_identity_providers** variable. OpenShift Container Platform supports the following identity providers:

- *Allow all*: The default identity provider when running OpenShift Container Platform without a master configuration file.
- *Deny all*: The default identity provider when using the quick or advanced installation method. It denies access for all user names and passwords.
- *LDAP Authentication*: This identity provider uses simple bind authentication to validate user names and passwords against an LDAPv3 server.
- *Request Header*: This identity provider identifies users from request header values, such as **X-Remote-User**. The request header is used in conjunction with an authenticating proxy, which configures the request header value.
- *Htpasswd*: The identity provider validates user names and passwords against a flat file generated using **htpasswd** command. For example, to generate the password **redhat** for the **admin** user with **htpasswd**, perform the following:

```
[root@demo ~]# htpasswd -nb admin redhat  
admin:$apr1$Y9pwjRCD$iqLLsX/vhUljsTx1Yq9FY.
```

Use the **openshift_master_identity_providers** variable in the Ansible inventory file to configure the identity provider.

```
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',  
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', ❶  
'filename': '/etc/origin/master/htpasswd'}]]❷
```

- ❶ The **HTPasswdPasswordIdentityProvider** is responsible for using the **htpasswd** command as the identity provider.
- ❷ The location of the credentials file.



References

Further information is available in the chapter on installing a cluster in the *OpenShift Container Platform 3.6 Installation and Configuration Guide* for Red Hat Enterprise Linux 7 at
<https://access.redhat.com/documentation/en-US/index.html>

Guided Exercise: Building the Ansible Inventory

In this exercise, you will create an inventory hosts file for the **mycluster.com** cluster and push it to the Git repository.

Outcomes

You should be able to:

- Clone a Git repository.
- Create a project skeleton, subdomain directory, and inventory hosts file for the **mycluster.com** cluster.
- Add Ansible cluster variables to the inventory hosts file.
- Add, commit, and push the inventory hosts file to a Git repository.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab build-inventory setup** command. The script ensures that all virtual machines are running and have network connectivity.

```
[student@workstation ~]$ lab build-inventory setup
```

Steps

1. Use the **ssh** command to log in to the **console** VM as the **root** user.

```
[student@workstation ~]$ ssh root@console
```

2. Clone the Ansible inventory file from Git.

- 2.1. From the **console** VM, use the **git clone** command in the **/root** directory to retrieve the Ansible reference inventory file.

```
[root@console ~]# git clone \
http://services.lab.example.com/openshift-install
```

- 2.2. In the **openshift-install** directory, create a project skeleton file called **openshift-cluster** and a subdomain directory called **mycluster.com**.

```
[root@console ~]# mkdir -p \
/root/openshift-install/openshift-cluster/mycluster.com
```

3. Create password hashes used for authentication to the web console for both the **admin** and **developer** users.

Use the **htpasswd** command with the **-nb** options to generate a password hash for the **admin** and **developer** users, using **redhat** as the password.

```
[root@console ~]# htpasswd -nb admin redhat
admin:$apr1$Y9pwjRCD$iqLLsX/vhUljsTxixYq9FY.
[root@console ~]# htpasswd -nb developer redhat
developer:$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751
```

Copy and paste both passwords in a file. You are going to use it in a later step.

4. Create the skeleton inventory hosts file **/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster**. The inventory file is based on the **mycluster** *Figure 2.1: Theoretical cluster diagram* located in the previous chapter.

```
# hosts file for mycluster.com
[OSEv3:children]
masters
etcd
nodes
nfs
lb
```

5. Add the cluster variables for networking, DNS, load-balancing, and authentication to the inventory file. Include the credentials generated in step *Step 3*.

Update the cluster variable values for networking, DNS, load balancing, node labels, and authentication with HTPasswd to the **/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster** inventory file. Include the password hashes for the users **admin** and **developer** created in the *Step 3*. An example host inventory file is located on the **console** virtual machine at **/root/D0380/labs/build-inventory/hosts.example** for reference.

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/
origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pwjRCD$iqLLsX/vhUljsTxixYq9FY.',
'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751'}
```

```
openshift_master_cluster_method=native
```

6. From **workstation**, open a new terminal and run the **lab build-inventory grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab build-inventory grade
...
Comparing Entries in [OSEv3:vars]

. Checking openshift_deployment_type..... PASS
. Checking deployment_type..... PASS
. Checking openshift_release..... PASS
. Checking osm_cluster_network_cidr..... PASS
. Checking openshift_portal_net..... PASS
...
```

7. Push the changes to Git.

- 7.1. Add the Ansible inventory file to the Git repository.

Go to the **/root/openshift-install** directory and add the project skeleton directory **openshift-cluster**, the subdomain directory **mycluster.com**, and the host inventory file **hosts-mycluster** to the Git repository.

```
[root@console ~]# cd /root/openshift-install
[root@console openshift-install]# git add \
  openshift-cluster/mycluster.com/hosts-mycluster
```

- 7.2. Commit the project skeleton directory **openshift-cluster** to the Git repository with the comment **Creation of the mycluster.com cluster**.

```
[root@console openshift-install]# git commit -m \
  'Creation of the mycluster.com cluster'
```

- 7.3. Use the **git push** command to send your changes to the Git repository.

```
[root@console openshift-install]# git push
Counting objects: 6, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 875 bytes | 0 bytes/s, done.
Total 5 (delta 0), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install
  daf18a6..ea64a1f  master -> master
```

This concludes this guided exercise.

Configuring the Integrated Registry

Objective

After completing this section, students should be able to configure the Ansible inventory file to request the installation of a persistent integrated registry.

OpenShift Container Registry

OpenShift Container Platform provides an integrated container registry called *OpenShift Container Registry* (OCR) that adds the ability to automatically provision new image repositories on demand. This provides users with a built-in location for their application builds to push the resulting images.

Whenever a new image is pushed to the OCR, the registry notifies OpenShift Container Platform about the new image, passing along all the information about it, such as the namespace, name, and image metadata. Different parts of OpenShift Container Platform react to new images, creating new builds, and deployments.

Persistent Registry Storage

At a high level, you can break down the OpenShift persistent storage support into two categories:

Block storage

Volumes or disks that can be mounted to only one container at a time (known as **ReadWriteOnce** mode). Examples of block storage are OpenStack Cinder, Ceph RBD, Amazon Elastic Block Storage, iSCSI, Fibre Channel. Most database technologies prefer block storage.

Shared file systems

Volumes that can be mounted for reading and writing by many containers simultaneously (known as **ReadWriteMany** mode). At the time of writing the only two available shared file systems supported are NFS and GlusterFS. Many legacy application runtimes prefer this type of storage for sharing data on disk.



Note

Most multi-tenant OpenShift deployments need to provide at least one persistent storage provider in each category in order to cover application use cases. In addition to application use cases, several of the core services that ship with OpenShift also require persistent volumes.

Persistent Storage Configuration Options

The Kubernetes persistent volume framework allows you to provision an OpenShift cluster with persistent storage using networked storage. This can be done after completing the initial OpenShift Container Platform installation depending on your application needs, giving users a way to request those resources without having any knowledge of the underlying infrastructure.

The following list represents examples of supported plug-ins for persistent volumes in OpenShift Container Platform:

NFS

OpenShift Container Platform clusters is provisioned with persistent storage using NFS. Persistent volumes (PVs) and persistent volume claims (PVCs) provide a convenient method for sharing a volume across a project. The NFS-specific information contained in a PV definition could also be defined directly in a pod definition, but this does not create the volume as a distinct cluster resource, which makes the volume more susceptible to conflicts.

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. To provision NFS volumes, a list of NFS servers and export paths are all that is required.

GlusterFS

There are two deployment solutions available when using Red Hat Gluster Storage, using either a containerized or dedicated storage cluster.

To provision GlusterFS volumes using the dedicated storage cluster solution, the following are required:

- An existing storage device in your underlying infrastructure
- A distinct list of servers (IP addresses) in the Gluster cluster, to be defined as endpoints
- An existing Gluster volume to be referenced in the persistent volume object
- The *glusterfs-fuse* package installed on each schedulable OpenShift Container Platform node in your cluster

OpenStack Cinder

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for OpenStack, all that is required for Cinder is a Cinder volume ID and the PersistentVolume API.

Ceph RBD

To provision Ceph volumes, the following are required:

- An existing storage device in your underlying infrastructure
- The Ceph key to be used in an OpenShift Container Platform secret object
- The Ceph image name
- The file system type used by block storage, for example, ext4
- The *ceph-common* package installed on each schedulable OpenShift Container Platform node in the cluster

AWS Elastic Block Store (EBS)

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift is configured for AWS Elastic Block Store (EBS), all that is required for OpenShift and AWS is an AWS EBS volume ID and the PersistentVolume API.

GCE Persistent Disk

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for GCE persistent disk, all that is required for OpenShift Container Platform and GCE is a GCE persistent disk volume ID and the PersistentVolume API.

iSCSI

Verify that the storage exists in the underlying infrastructure before mounting it as a volume in OpenShift Container Platform. All that is required for OpenShift Container Platform and iSCSI is the iSCSI target portal, a valid iSCSI Qualified Name (IQN), a valid *logical unit number* (LUN), the file-system type, and the PersistentVolume API.

Fibre Channel

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. All that is required for Fibre Channel persistent storage is the target WWNs (array of fibre channel target's world wide names), a valid LUN, and file system type, and the PersistentVolume API. Note, the number of LUNs must correspond to the number of persistent volumes that are created.

Azure Disk

Storage must exist in the underlying infrastructure before it can be mounted as a volume in OpenShift Container Platform. After ensuring OpenShift Container Platform is configured for Azure Disk, all that is required for OpenShift Container Platform and Azure is an Azure Disk Name and Disk URI and the PersistentVolume API.

Registry Configuration

An OpenShift registry is created during installation if there are nodes present with labels matching the default registry selector, **region=infra**. To label nodes, set **openshift_node_labels** per node as needed in the Ansible host inventory file.

```
[nodes]
node.example.com openshift_node_labels="{'region': 'infra'}"
```

Registry Storage Variable Options

NFS Host Group

An NFS volume is created with the path **nfs_directory/volume_name** on the host in the **[nfs]** host group. For example, the volume path using these options is **/exports/registry**.

```
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
```

External NFS Host

NFS volume must already exist with path **nfs_directory/_volume_name** on the storage_host. For example, the remote volume path using these options is **nfs.example.com:/exports/registry**.

```
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_host=nfs.example.com
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
```

Openstack

The volume must already exist.

```
openshift_hosted_registry_storage_kind=openstack
openshift_hosted_registry_storage_access_modes=['ReadWriteOnce']
openshift_hosted_registry_storage_openstack_filesystem=ext4
openshift_hosted_registry_storage_openstack_volumeID=3a650b4f-c8c5-4e0a-8ca5-
eaae11f16c57
openshift_hosted_registry_storage_volume_size=10Gi
```

AWS S3

The AWS S3 bucket must already exist.

```
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
openshift_hosted_registry_storage_s3_encrypt=false
openshift_hosted_registry_storage_s3_kmskeyid=aws_kms_key_id
openshift_hosted_registry_storage_s3_accesskey=aws_access_key_id
openshift_hosted_registry_storage_s3_secretkey=aws_secret_access_key
openshift_hosted_registry_storage_s3_bucket=bucket_name
openshift_hosted_registry_storage_s3_region=bucket_region
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
```

Any S3 Service Provider

In addition to the AWS S3 bucket that must already exist, the region end points must also be configured for S3 service provider entries.

```
openshift_hosted_registry_storage_kind=object
openshift_hosted_registry_storage_provider=s3
openshift_hosted_registry_storage_s3_accesskey=access_key_id
openshift_hosted_registry_storage_s3_secretkey=secret_access_key
openshift_hosted_registry_storage_s3_regionendpoint=https://myendpoint.example.com/
openshift_hosted_registry_storage_s3_bucket=bucket_name
openshift_hosted_registry_storage_s3_region=bucket_region
openshift_hosted_registry_storage_s3_chunksize=26214400
openshift_hosted_registry_storage_s3_rootdirectory=/registry
openshift_hosted_registry_pullthrough=true
openshift_hosted_registry_acceptschema2=true
openshift_hosted_registry_enforcequota=true
```

CloudFront

When using CloudFront, all three of the following variables must be defined.

```
openshift_hosted_registry_storage_s3_cloudfront_baseurl=https://
myendpoint.cloudfront.net/
openshift_hosted_registry_storage_s3_cloudfront_privatekeyfile=/full/path/to/
secret.pem
openshift_hosted_registry_storage_s3_cloudfront_keypairid=yourpairid
```

GlusterFS

The default behavior is to install both the **ceph** and **glusterfs** plug-in dependencies, if available.

```
osn_storage_plugin_deps=['ceph', 'glusterfs']
```

When using GlusterFS there are two inventory file configurations available, one for a new cluster deployment with GlusterFS and the other for GlusterFS storage on an existing cluster.

Sizing Registry Storage

Sizing considerations for the registry depend on several factors including an understanding of how images are processed.

The following list presents examples of external registries.

- Docker registry
- Red Hat registry
- Private registry

Images stored in an external registry are compressed and not expanded until they are pulled and made available for use to start a pod. When an image is pulled, such as, using the **oc new-app** command, it pulls the image from an external registry to a local node registry where the image is expanded and available for use. The images stored in the local node registry can either be deleted once the pod is deployed or kept for future use.

The size of an external registry is based on a total number of expected images. Each image is an immutable collection of layers that are compressed into a single image. It is hard to size an external registry because multiple images can share layers. Changes to the image adds more layers. Plus, some layers may be shared with other images. Therefore, a direct sizing of an image is extremely challenging when using it to determine the size of the registry.

The size of the local node registry is based on how many expanded images are expected to remain in local storage for future use. In a small deployment you may find 10 GB of registry storage for an expected number of images sufficient. However, the size of an image is not reflective of what the calculation should be for registry storage. The total number of images and the length of time an image is expected to remain in storage are better calculation choices when sizing registry storage.

Types of Registries

Public external registry

Docker and Red Hat registries are both examples of shared registries that are outside the firewall. Docker and Red Hat are responsible for sizing these registries.

Private external registry

A stand-alone private registry is also a shared registry and although it is external to OpenShift it is located inside the firewall. The size of a private external registry is based on the total number of expected images. Each image is an immutable collection of layers that are compressed into a single image. It is hard to size a private registry because multiple images can share layers. Changes to the image add more layers. Further, some layers may be shared with other images. Therefore, an exact sizing of an image is extremely challenging when trying to determine the size of the registry.

Internal registry

An internal registry is created during the installation of OpenShift and runs in a container. This is where images are stored that are not for external consumption, and is only to be used by nodes in that cluster for instantiating pods.

Local Node Storage

Located on each application node, this ephemeral storage is where images are stored that are pulled from a registry. These images are used to deploy containers and pods. Images stored here are either deleted after the container or pod is created or kept for future use. If images in local node storage are not deleted, they can be used to deploy additional containers or pods. If they are deleted they are regenerated the next time the image is pulled from one of the registries.

Stand-alone Registry

Instead of deploying OpenShift Container Platform as a full PaaS environment for developers with an integrated registry, you can install OpenShift Container Registry as a stand-alone container registry to run on-premise or in the cloud.

When installing a stand-alone deployment of the registry, a cluster of masters and nodes is still installed, similar to a typical OpenShift Container Platform installation. Then, the container registry is deployed to the cluster. This stand-alone deployment option is typical in disconnected OpenShift environments and is useful for administrators who want a container registry, but do not require the full OpenShift Container Platform environment that includes the developer-focused web console and application build and deployment tools.



References

Further information is available in the chapter on installing a stand-alone deployment of OpenShift Container Registry in the *Installation and Configuration Guide* for OpenShift Container Platform 3.6 at

https://access.redhat.com/documentation/en-US/index.html

Guided Exercise: Configuring the Integrated Registry

In this guided exercise, you will configure the Ansible inventory file to request the installation of a persistent integrated registry.

Outcomes

You should be able to:

- Update an Ansible inventory file for the configuration of an OpenShift Container Platform integrated registry using a NFS back end.
- Update a Git working copy using **git** commands.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Before you begin

The guided exercise from *the section called “Guided Exercise: Building the Ansible Inventory”* must be completed. If not, complete the guided exercise before running this exercise.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab int-registry setup** command. The script ensures that the cluster hosts are reachable and checks for the presence of the **/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster** Ansible inventory file on the **console** VM.

```
[student@workstation ~]$ lab int-registry setup
```

Steps

1. On the **console** VM, update your Git working copy by retrieving any updates available from the remote Git repository.
 - 1.1. On the **workstation** VM, open a new terminal and use the **ssh** command to log in to **console** as the **root** user. Ensure that the **/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster** inventory file exists.

```
[student@workstation ~]$ ssh root@console
[root@console ~]# ls /root/openshift-install/openshift-cluster/mycluster.com
hosts-mycluster
```



Note

The inventory file was created in the section called “*Guided Exercise: Building the Ansible Inventory*”. A copy of the inventory file is available at **/root/D0380/solutions/int-registry/openshift-cluster/mycluster.com/hosts-mycluster** on the **console** VM.

- 1.2. Go to the **/root/openshift-install/** directory and run **git pull** to retrieve the latest changes.

```
[root@console ~]# cd /root/openshift-install
[root@console openshift-install]# git pull
Already up-to-date.
```

2. Add the inventory variables for the configuration of the OpenShift Container Platform integrated registry.
- 2.1. The classroom environment has images that are about 250 MiB in size, and an average of 400 images will be deployed.

Allocate 10 GiB for the volume used by the integrated registry in an NFS. The variable that defines the volume size is **openshift_hosted_registry_storage_volume_size**.

Use a text editor to edit the **hosts-mycluster** inventory file and append the following variables, in the **[OSEv3:vars]** section:

```
openshift_hosted_registry_storage_kind=nfs ①
openshift_hosted_registry_storage_access_modes=['ReadWriteMany'] ②
openshift_hosted_registry_storage_nfs_directory=/exports ③
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)' ④
openshift_hosted_registry_storage_volume_name=registry ⑤
openshift_hosted_registry_storage_volume_size=10Gi ⑥
```

- ① Appends the following variable, which enables NFS as the back end.
- ② Adds the variable that defines the access mode, **ReadWriteMany**, which allows the volume to be mounted as **read/write** by many nodes.
- ③ Adds the variable that defines the name of the exports directory.
- ④ Adds the variable that defines the NFS storage mode.
- ⑤ Adds the variable that defines the registry volume.
- ⑥ Adds the variable that defines the registry volume size.

- 2.2. The updated inventory should read as follows:

```
# hosts file for mycluster.com
[OSEv3:children]
masters
```

```

etcd
nodes
nfs
lb

[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{"name": "htpasswd_auth", "login": "true",
"challenge": "true", "kind": "HTPasswdPasswordIdentityProvider", "filename": "/etc/origin/master/htpasswd"}]
openshift_master_htpasswd_users={"admin": "$apr1$Y9pwjRCD$iqLLsX/vhUljsTxiYq9FY.", "developer": "$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751"}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi

```

- From **workstation**, open a new terminal and run the **lab int-registry grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```

[student@workstation ~]$ lab int-registry grade
...
Comparing Entries in [OSEv3:vars]

· Checking openshift_deployment_type..... PASS
· Checking deployment_type..... PASS
· Checking openshift_release..... PASS
· Checking osm_cluster_network_cidr..... PASS
· Checking openshift_portal_net..... PASS
...

```

- Commit your changes to the inventory by running the **git commit** command.

```

[root@console openshift-install]# git commit . -m \
'Adds the integrated registry configuration'

```

- Push your changes to the remote repository.

```

[root@console openshift-install]# git push
Counting objects: 9, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 564 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install

```

```
ea64a1f..5678d94 master -> master
```

This concludes the guided exercise.

Preparing a Disconnected Environment

Objective

After completing this section, students should be able to configure the Ansible inventory and cluster environment to run disconnected from the Internet.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

OpenShift Disconnected

Frequently, portions of a data center may not have access to the Internet, even through proxy servers. Installing OpenShift Container Platform in these environments is considered a disconnected installation.

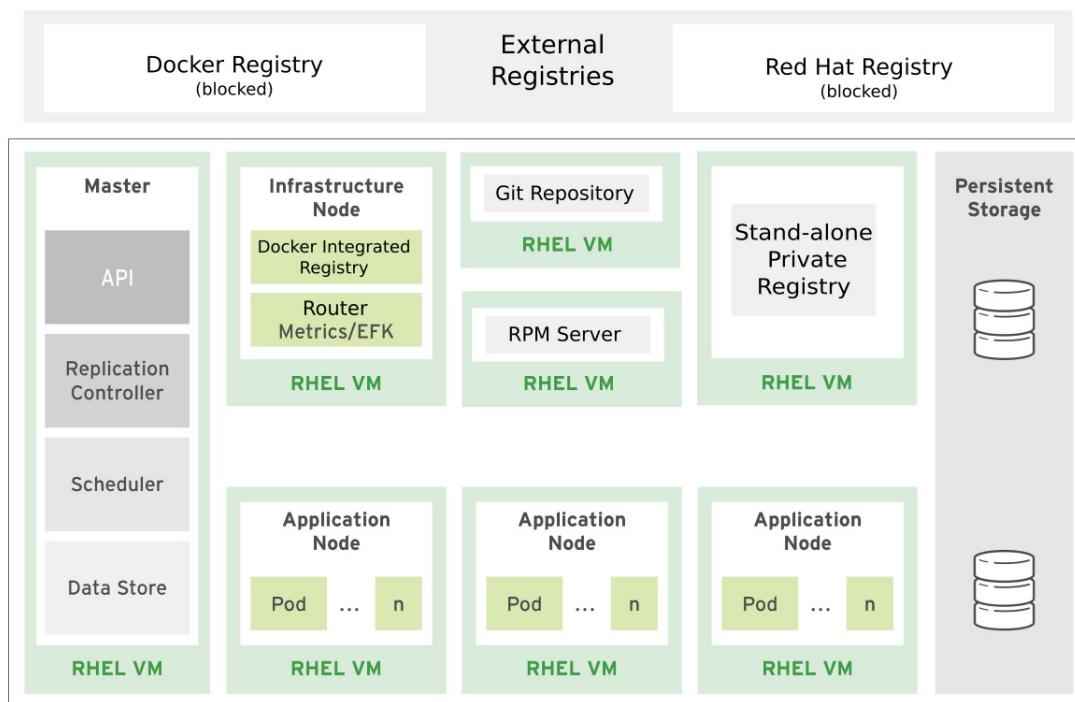


Figure 2.2: OpenShift disconnected cluster architecture

An OpenShift Container Platform disconnected installation differs from a regular installation in two primary ways:

- The OpenShift Container Platform software channels and repositories are not available through the Red Hat content distribution network.

- OpenShift Container Platform uses several containerized components. Normally, these images are pulled directly from Red Hat's Docker registry. In a disconnected environment, this is not possible.

A disconnected installation ensures the OpenShift Container Platform software is available to the relevant servers, and then follows the same installation process as a standard connected installation.

After you have installed OpenShift Container Platform, you need source code in a source control repository, for example, Git, before you can use it.

Additionally, when building applications in OpenShift Container Platform, your build may have some external dependencies, such as a Maven repository or Gem files for Ruby applications. Consequently, and because they might require certain tags, many of the quickstart templates offered by OpenShift Container Platform may not work in a disconnected environment. However, while Red Hat container images try to reach out to external repositories by default, you can configure OpenShift Container Platform to use your own internal repositories.



Note

You can also have a Red Hat Satellite Server that provides access to Red Hat content through an intranet or LAN. For environments with Satellite, you can synchronize the OpenShift Container Platform software with the Satellite for use with the OpenShift Container Platform servers.

Red Hat Satellite 6.1 also introduces the ability to act as a Docker registry, and it can be used to host the OpenShift Container Platform containerized components. Doing so is outside of the scope of this course.

Required Software and Components

To access and download the required software repositories and container images, you will need a Red Hat Enterprise Linux (RHEL) 7 server with access to the Internet and at least 100 GiB of additional free space.

Yum Repositories

The most common way to install the OpenShift Container Platform is to use RPM packages. The following Yum repositories are required:

- rhel-7-fast-datapath-rpms
- rhel-7-server-extras-rpms
- rhel-7-server-optional-rpms
- rhel-7-server-ose-3.6-rpms

There are several options for synchronizing RPM content internally:

- Synchronizing subscription-manager channels through Satellite 6
- Creating and synchronizing custom channels through Satellite 5
- Creating and synchronizing a custom RPM server

Building an RPM Repository Server

To build a separate server to act as the RPM repository server, install a new RHEL 7 system with at least 110 GB of space, and select the Basic Web Server group option during the installation.

- If necessary, attach the external storage, and then copy the repository files into the Apache root folder:

```
# cp -a /path/to/repos /var/www/html/
# chmod -R +r /var/www/html/repos
# restorecon -vR /var/www/html
```

- Add the firewall rules:

```
# firewall-cmd --permanent --add-service=http
# firewall-cmd --reload
```

- Enable and start Apache for the changes to take effect:

```
# systemctl enable httpd
# systemctl start httpd
```

Creating a Stand-alone Private Registry

During installation, OpenShift pulls images from Red Hat in order to provide services such as the integrated registry, router, and base images for pods, S2I builds, and others. In most cases, access to the Red Hat public registry is blocked or restricted by web proxies. A stable and long-term solution to this restriction is to create a standalone registry and populate it with the images that OpenShift requires. You can then point OpenShift to your standalone registry to retrieve the necessary images. This allows you to establish a much simpler and more automated process for updating images when needed.

Create a Private Registry

To create a private registry, install a new registry on a RHEL 7 server, then synchronize images to the newly created private registry. To do this, you need a host that has access to external registries, such as the Docker and Red Hat registries, has the **docker** service installed, and has access to the private Docker registry host, such as **registry.mycluster.example.com:5000**.

The process of creating the registry is as follows:

1. Install the *docker*, *docker-distribution*, and *firewalld* packages:

```
# yum install -y docker docker-distribution firewalld
```

2. Enable and start the **firewalld** service:

```
# systemctl enable firewalld
# systemctl start firewalld
```

3. Open TCP port 5000 on the firewall to allow access to the private registry:

```
# firewall-cmd --add-port 5000/tcp --permanent  
# firewall-cmd --reload
```

4. Enable and start the **docker-distribution** service:

```
# systemctl enable docker-distribution  
# systemctl start docker-distribution
```

The following commands are examples that confirm connectivity to both the Red Hat registry and the newly installed private registry.

- Test your connection to the Red Hat registry:

```
$ curl -IL registry.access.redhat.com  
HTTP/1.0 302 Found  
Location: https://access.redhat.com/search/#/container-images  
Server: BigIP  
Connection: close  
Content-Length: 0  
  
HTTP/2 200  
...
```

- Test the connection to your private registry:

```
$ curl -I registry.mycluster.example.com:5000  
HTTP/1.1 200 OK  
Cache-Control: no-cache  
Date: Mon, 10 Apr 2017 15:18:09 GMT  
Content-Type: text/plain; charset=utf-8
```

Configure Docker on Each Host

On each host in the OpenShift cluster, configure the internal registry as an insecure registry by adding the following line to the **/etc/sysconfig/docker** file. The **INSECURE_REGISTRY** option instructs the Docker daemon to trust any Docker registry on the indicated subnet, rather than requiring a certificate.

```
INSECURE_REGISTRY='--insecure-registry registry.mycluster.com:5000'
```

Restart the **docker** service:

```
# systemctl restart docker
```

Synchronize External Images with the Integrated Docker Registry

The following examples use **curl** to download the script and configuration files, to download the images from an external registry, and synchronize them with a private registry:

```
# curl -O https://raw.githubusercontent.com/redhat-cop/openshift-toolkit/master/disconnected_registry/docker-registry-sync.py

# curl -O https://raw.githubusercontent.com/redhat-cop/openshift-toolkit/master/disconnected_registry/docker_tags.json

# chmod +x docker-registry-sync.py

# ./docker-registry-sync.py \
--from=registry.access.redhat.com --to=registry.mycluster.example.com:5000 \
--file=./docker_tags.json --openshift-version=3.6
```

Modify the Ansible Inventory File

Add the following entries to the Ansible inventory file to point OpenShift to your private registry, and disable access to the default external registries.

```
openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io
oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true
```

Listing and Searching Images from the Docker Registry

The **docker-registry-cli** command is a utility that allows you to search images in any Docker registry. It is a wrapper script calling an open source Python application that can query Docker registries.

Run the command with the URL of the registry, including the access port, as the argument. For example, **registry.mycluster.example.com:5000**. The **list all** option allows you to list all the images present in the registry.

```
$ docker-registry-cli \
registry.mycluster.example.com:5000 list all

-----
1) Name: openshift3/ose-haproxy-router
Tags: v3.6.173.0.49
-----
2) Name: openshift3/registry-console
Tags: v3.6
...
13 images found !
```

Use the **search** option and the name of the image to search for a particular image.

```
$ docker-registry-cli \
registry.mycluster.example.com:5000 search openshift3/ose

-----
1) Name: openshift3/ose
Tags: v3.6.173.0.49
1 images found !
```

Image Streams

An image stream comprises any number of Docker-formatted container images identified by tags. An image stream tag is a named pointer to an image in an image stream and can reference any local or externally managed image. A tagged image stream presents a single virtual view of related images, similar to an image repository, and may contain images from any of the following:

- Its own image repository in the OpenShift Container Platform integrated registry
- Image repositories from external registries

The image streams from OpenShift are automatically updated by the installer's playbook as long as the Ansible inventory file is updated according to *the section called "Modify the Ansible Inventory File"*.

References

Docker registry CLI application
https://github.com/vivekjuneja/docker_registry_cli/

Further information is available in the chapter on installing a cluster in the *OpenShift Container Platform 3.6 Installation and Configuration Guide* for Red Hat Enterprise Linux 7 at
<https://access.redhat.com/documentation/en-US/index.html>

Guided Exercise: Preparing a Disconnected Environment

In this exercise, you will update an Ansible inventory file to prepare a disconnected environment.

Outcomes

You should be able to:

- Query the Git repository using Git commands.
- Interact with a Docker registry using the **docker-registry-cli** command-line interface.
- Update the Ansible inventory file for a Docker integrated registry.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Before you begin

The exercises from the section called “*Guided Exercise: Building the Ansible Inventory*” and the section called “*Guided Exercise: Configuring the Integrated Registry*” must have been completed. If not, complete the guided exercises before running this exercise.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab offline setup** command. The script ensures that all virtual machines are running and have network connectivity. The script also downloads the required material for this exercise.

```
[student@workstation ~]$ lab offline setup
```

Steps

1. Ensure that the Ansible inventory file is up-to-date in the **console** VM.

- 1.1. Access the **console** VM.

On the **workstation** VM, open a new terminal and use the **ssh** command to log in to the **console** VM as the **root** user.

```
[student@workstation ~]$ ssh root@console
```

- 1.2. On the **console** VM ensure that the inventory file is present in the **/root/openshift-install/openshift-cluster/mycluster.com** directory.

Navigate to the **/root/openshift-install/openshift-cluster/mycluster.com** directory. Ensure that the **hosts-mycluster** inventory file, created in the previous exercise, is present in the directory.

```
[root@console ~]# cd /root/openshift-install/openshift-cluster/mycluster.com
[root@console mycluster.com]# ls
hosts-mycluster
```

- 1.3. Ensure that the last guided exercise Ansible inventory file is available.

Use a text editor to inspect the inventory file and compare it to the following output:

```
# hosts file for mycluster.com
[OSEv3:children]
masters
etcd
nodes
nfs
lb

[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{"name": "htpasswd_auth", "login": "true",
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pwjRCD$iqLLsX/vhUljsTxiYq9FY.', 'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET0lv751'}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
```

If the file does not match the output, update the file.



Note

Optionally, use the inventory file located in the **/root/D0380/solutions/offline-env/** directory as the base file.

- 1.4. Run the **git pull** command to ensure that the latest changes are present in the inventory file.

```
[root@console mycluster.com]# git pull
Already up-to-date.
```

2. Review the Yum repositories configured on the **console** VM.

2.1. List the Yum repository files.

Go to the **/etc/yum.repos.d** directory and list the files.

```
[root@console mycluster.com]# cd /etc/yum.repos.d
[root@console yum.repos.d]# ls
redhat.repo          rhel-7-server-optional-rpms.repo  rhel_dvd.repo
rhel-7-fast-datapath-rpms.repo  rhel-7-server-ose-3.6-rpms.repo
rhel-7-server-extras-rpms.repo  rhel-7-server-updates-rpms.repo
```

2.2. Retrieve the offline repository entries.

Use the **grep** command to inspect the URL for the repository files. The various entries define the repositories that contain the OpenShift Container Platform packages for offline installation. The content server, **content.example.com**, stores the installation packages.

```
[root@console yum.repos.d]$ grep baseurl *.repo
rhel-7-fast-datapath-rpms.repo:baseurl = http://content.example.com/ocp3.6/
x86_64/fast
rhel-7-server-extras-rpms.repo:baseurl = http://content.example.com/ocp3.6/
x86_64/extras
rhel-7-server-optional-rpms.repo:baseurl = http://content.example.com/ocp3.6/
x86_64/rhelopt
rhel-7-server-ose-3.6-rpms.repo:baseurl = http://content.example.com/ocp3.6/
x86_64/ocp
rhel-7-server-updates-rpms.repo:baseurl = http://content.example.com/ocp3.6/
x86_64/updates
rhel_dvd.repo:baseurl = http://content.example.com/rhel7.4/x86_64/dvd
```



Note

The **rhel-7-* .repo** repository files are the same on all hosts that comprise the environment.

2.3. List the repositories on the system.

Run the **yum repolist** command from the **console** VM.



Note

If an exclamation mark is displayed in front of a repository name, it means that it has expired. Run the **yum clean all** command to refresh the repositories.

```
[root@console yum.repos.d]# yum repolist
Loaded plugins: langpacks, search-disabled-repos
repo id
repo name
status
```

```

!rhel-7-fast-datapath-rpms
    Remote classroom copy of Fast Datapath RPMS
        16

!rhel-7-server-extras-rpms
    Remote classroom copy of RHEL Extras RPMS
        100

!rhel-7-server-optional-rpms
    Remote classroom copy of RHEL Optional RPMS
        4,848

!rhel-7-server-ose-3.6-rpms
    Remote classroom copy of OCP RPMS
        483

!rhel-7-server-updates-rpms
    Remote classroom copy of RHEL Updates
        574

!rhel_dvd
    Remote classroom copy of dvd
        4,986

repolist: 11,007

```

3. Add the variables that configure the Docker integrated registry.

Update the **hosts-mycluster** Ansible inventory file to include the references to an external Docker integrated registry, and commit your changes.

- 3.1. On the **console** VM, go to the **/root/openshift-install/openshift-cluster/mycluster.com** directory and use a text editor to open the Ansible inventory file.

Append the variables in the **[OSEv3:vars]** group:

```

openshift_docker_additional_registries=registry.mycluster.com:5000 ①
openshift_docker_insecure_registries=registry.mycluster.com:5000 ②
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io ③
oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version} ④
openshift_examples_modify_imagestreams=true ⑤

```

- ① Declares additional registries.
- ② Declares the registry as insecure.
- ③ Prevents users from accessing the registries available at **registry.access.redhat.com** and **docker.io**.
- ④ Defines the alternative location for images. The variable is necessary if you are not using the default registry at **registry.access.redhat.com**.
- ⑤ Set the value to **true** if you are using a registry other than the default registry. The value modifies the image stream location to the value defined by **oreg_url**.

The Ansible inventory file should read as follows:

```

# hosts file for mycluster.com
[OSEv3:children]
masters
etcd
nodes
nfs
lb

[OSEv3:vars]

```

```

openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{"name": "htpasswd_auth", "login": "true",
"challenge": "true", "kind": "HTPasswdPasswordIdentityProvider", "filename": '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={"admin": "$apr1$Y9pwjRCD$iqLLsX/vhUljsTxiYq9FY.", "developer": "$apr1$AH9Loha.$GJNmEjjKjt/8LPET0lv751"}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi

openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com,docker.io
oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

```

- 3.2. From **workstation**, open a new terminal and run the **lab offline grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```

[student@workstation ~]$ lab offline grade
...
Comparing Entries in [OSEv3:vars]

. Checking openshift_deployment_type..... PASS
. Checking deployment_type..... PASS
. Checking openshift_release..... PASS
. Checking osm_cluster_network_cidr..... PASS
. Checking openshift_portal_net..... PASS
...

```

- 3.3. From the **console** VM, save the file and add your changes to Git.

```
[root@console mycluster.com]# git add .
```

- 3.4. Commit the updated inventory file to the remote Git repository. Use **Adds offline settings** as the comment.

```
[root@console mycluster.com]# git commit -m \
'Adds offline settings'
```

- 3.5. Push the changes to the Git repository.

```
[root@console mycluster.com]# git push
Counting objects: 9, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (5/5), 560 bytes | 0 bytes/s, done.
Total 5 (delta 2), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install
  5678d94..a61e7b1 master -> master
```

3.6. Exit from the **console** VM.

4. Review the Git repository where the Ansible Playbook is stored on the **services** VM.

4.1. List the available Git repositories.

On the **workstation** VM, open a web browser and navigate to **registry.lab.example.com**.

Project	Description	Owner	Last Change
openshift-install	Unnamed repository; edit this...	Apache	22 min ago
php-helloworld	Unnamed repository; edit this...	Apache	4 weeks ago
temps	Unnamed repository; edit this...	Apache	4 weeks ago

Figure 2.3: The Git repository available at services.lab.example.com

4.2. Click the **openshift-install** repository to access the history.

summary | [shortlog](#) | [log](#) | [commit](#) | [commitdiff](#) | [tree](#)

shortlog

Date	Author	Message	Commit Type
2 min ago	root	Adds offline settings	master
2 hours ago	root	Adds the integrated registry configuration	commit
6 hours ago	root	Creation of the mycluster.com cluster	commit
2017-11-24	root	First commit	commit
2017-11-21	root	Initial commit	commit
2017-11-12	root	Establish remote repository	commit

heads

Branch	Shortlog	Log	Commit	Tree
master	shortlog	log	commit	tree

Unnamed repository; edit this file 'description' to name the repository.

Figure 2.4: The openshift-install repository



Note

The commits messages may be different.

5. Browse the images from the Docker registry running on the **services** VM.

On the **workstation** VM, open a terminal and use the **docker-registry-cli** command to browse the images from the Docker registry running on the **services** server. A DNS entry, **registry.lab.example.com**, points to the VM.

- 5.1. List all the images in the **services** VM registry.

The **docker-registry-cli** is a utility installed on the **workstation** and **console** VM that allows you to access insecure Docker registries.

Run the command with the URL of the registry as the argument, **registry.lab.example.com:5000**. The **list all** option allows you to list all the images present in the registry.

```
[student@workstation ~]$ docker-registry-cli \
registry.lab.example.com:5000 list all

-----
1) Name: openshift3/ose-haproxy-router
Tags: v3.6.173.0.49
-----
2) Name: openshift3/registry-console
Tags: v3.6
...output omitted...

28 images found !
```

- 5.2. Use the **search** option to look for a particular image.

```
[student@workstation ~]$ docker-registry-cli \
registry.lab.example.com:5000 search openshift3/ose

-----
1) Name: openshift3/ose
Tags: v3.6.173.0.49

1 images found !
```

This concludes the guided exercise.

Lab: Preparing to Install an HA Cluster

In this lab, you will configure the Ansible inventory file to install OpenShift Container Platform.

Outcomes

You should be able to:

- Use **git** commands to retrieve an inventory file.
- Populate the Ansible inventory file with the required variables for an HA installation.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Before you begin

The workshops *Building the Ansible Inventory*, *Configuring the Integrated Registry*, and *Preparing a Disconnected Environment* must have been completed. If not, complete them before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab prep-install setup** command. The script installs the required files for this exercise.

```
[student@workstation ~]$ lab prep-install setup
```

Steps

1. Retrieve the latest changes from the Git repository on the **console** VM.

From the **console** VM, go to the **/root/openshift-install** directory and pull the source code from **services.lab.example.com** using in the **console** VM.

2. Update the Ansible inventory file to configure the infrastructure node's selectors.
 - The **openshift_router_selector** value defines the selector for the router pods.
 - The **openshift_registry_selector** value defines the selector for the registry pods.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
openshift_router_selector	region=infra
openshift_registry_selector	region=infra

3. Update the Ansible inventory file to configure the node traffic interface.

Update the **[OSEv3:vars]** group to add the **openshift_set_node_ip** value to change the node's traffic interface.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
<code>openshift_set_node_ip</code>	<code>True</code>

4. Update the Ansible inventory file to disable checks due to the hardware environment limitations.

Update the **[OSEv3:vars]** group to add the **openshift_disable_check** option to disable disk availability, docker storage, and memory availability.

- The **openshift_disable_check** option specifies that some checks not be performed.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
<code>openshift_disable_check</code>	<code>disk_availability,</code> <code>docker_storage,</code> <code>memory_availability,</code> <code>docker_image_availability</code>

5. Update the Ansible inventory file to use the **root user** to connect to remote servers and override the console image prefix.

Update the **[OSEv3:vars]** group to add the following configuration:

- The **openshift_cockpit_deployer_prefix** value overrides the console image prefix for enterprise deployments.
- The **ansible_ssh_user** value defines the user that Ansible uses for the connection to the nodes.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
<code>ansible_ssh_user</code>	<code>root</code>
<code>openshift_cockpit_deployer_prefix</code>	<code>'openshift3/'</code>

6. Declare the NFS group in the Ansible inventory file that declares the **console** VM as the NFS server.

In the **hosts-mycluster** inventory file, add an **nfs** group for the nodes that run the Docker registry. Give it a value of **nfs.mycluster.com**.

7. Set the RPM method as the installation method instead of running a containerized version of OpenShift Container Platform.
8. Create the **etcd** group and define the **etcd[1:5].mycluster.com** nodes as the ones that run an **Etd** server.
9. Customize the inventory file to identify the hosts by IP addresses.

On the **console** VM, navigate to the **/root/D0380/labs/prep-install/openshift-cluster/mycluster.com/** directory and review the **hosts-mycluster.fixme** file.

The file contains extra settings for the inventory file. Append the content of the file to the **hosts-mycluster** inventory file.

10. From the **/root/openshift-install/openshift-cluster/mycluster.com** directory, push the inventory file to your local Git repository. Use **Ansible HA master file** as the commit message.
11. Grade your work.

Run the **lab prep-install grade** script from the **workstation** VM to verify your work.

```
[student@workstation ~]$ lab prep-install grade
```

This concludes this lab.

Solution

In this lab, you will configure the Ansible inventory file to install OpenShift Container Platform.

Outcomes

You should be able to:

- Use **git** commands to retrieve an inventory file.
- Populate the Ansible inventory file with the required variables for an HA installation.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Before you begin

The workshops *Building the Ansible Inventory*, *Configuring the Integrated Registry*, and *Preparing a Disconnected Environment* must have been completed. If not, complete them before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab prep-install setup** command. The script installs the required files for this exercise.

```
[student@workstation ~]$ lab prep-install setup
```

Steps

1. Retrieve the latest changes from the Git repository on the **console** VM.

From the **console** VM, go to the **/root/openshift-install** directory and pull the source code from **services.lab.example.com** using in the **console** VM.

- 1.1. From the **workstation** VM, open a terminal and use the **ssh** command to log in to the **console** VM as the **root** user. Go to the **openshift-install** directory.

```
[student@workstation ~]$ ssh root@console
[root@console ~]# cd openshift-install
```

- 1.2. Retrieve the latest changes from the Git repository running on **services.lab.example.com**.

```
[root@console openshift-install]# git pull
Already up-to-date.
```

2. Update the Ansible inventory file to configure the infrastructure node's selectors.

Open the **/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster** Ansible inventory file, and update the **[OSEv3:vars]** group according to the following values:

- The **openshift_router_selector** value defines the selector for the router pods.
- The **openshift_registry_selector** value defines the selector for the registry pods.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
openshift_router_selector	region=infra
openshift_registry_selector	region=infra

- 2.1. Use a text editor to open the **hosts-mycluster** inventory file and add the variables mentioned in the table to the **OSEv3:vars** group.

The **[OSEv3:vars]** group should read as follows:

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pwjRCD$iqLLsX/vhUljsTxiYq9FY.', 'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751'}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi

openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com,docker.io

oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'
```

3. Update the Ansible inventory file to configure the node traffic interface.

Update the **[OSEv3:vars]** group to add the **openshift_set_node_ip** value to change the node's traffic interface.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
<code>openshift_set_node_ip</code>	<code>True</code>

- 3.1. The **[OSEv3:vars]** group should read as follows:

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pwjRCD$iqLLsX/vhUljsTxiYq9FY.', 'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751'}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi

openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io
oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'

openshift_set_node_ip=True
```

4. Update the Ansible inventory file to disable checks due to the hardware environment limitations.

Update the **[OSEv3:vars]** group to add the **openshift_disable_check** option to disable disk availability, docker storage, and memory availability.

- The **openshift_disable_check** option specifies that some checks not be performed.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
<code>openshift_disable_check</code>	<code>disk_availability,</code> <code>docker_storage,</code> <code>memory_availability,</code> <code>docker_image_availability</code>

4.1. The **[OSEv3:vars]** group should read as follows:

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pwjRCD$iqLLsX/vhUljsTxiYq9FY.', 'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751'}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi

openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io
oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'

openshift_set_node_ip=True
openshift_disable_check=disk_availability,docker_storage,memory_availability,\
docker_image_availability
```

- Update the Ansible inventory file to use the **root user** to connect to remote servers and override the console image prefix.

Update the **[OSEv3:vars]** group to add the following configuration:

- The **openshift_cockpit_deployer_prefix** value overrides the console image prefix for enterprise deployments.
- The **ansible_ssh_user** value defines the user that Ansible uses for the connection to the nodes.

The following table lists the values for the **OSEv3:vars** group.

Variable	Value
ansible_ssh_user	root
openshift_cockpit_deployer_prefix	'openshift3'

5.1. The **[OSEv3:vars]** group should read as follows:

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pjRCD$iqLLsX/vhUljsTxiYq9FY.', 'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET0lv751'}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi

openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io
oreg_url=registry.mycluster.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'

openshift_set_node_ip=True
openshift_disable_check=disk_availability,docker_storage,memory_availability\
docker_image_availability
ansible_ssh_user=root
openshift_cockpit_deployer_prefix='openshift3/'
```

6. Declare the NFS group in the Ansible inventory file that declares the **console** VM as the NFS server.

In the **hosts-mycluster** inventory file, add an **nfs** group for the nodes that run the Docker registry. Give it a value of **nfs.mycluster.com**.

- 6.1. In the **hosts-mycluster** inventory file, add the **nfs** group with a value of **nfs.mycluster.com**.

```
[nfs]
nfs.mycluster.com
```

7. Set the RPM method as the installation method instead of running a containerized version of OpenShift Container Platform.

- 7.1. In the **hosts-mycluster** inventory file, create the **1b** group and give it the value of **1b.mycluster.com containerized=false**.

```
[1b]
1b.mycluster.com containerized=false
```

8. Create the **etcd** group and define the **etcd[1:5].mycluster.com** nodes as the ones that run an **Etcd** server.

- 8.1. In the **hosts-mycluster** inventory file, add the **etcd** group with a value of **etcd[1:5].mycluster.com**.

```
[etcd]
etcd[1:5].mycluster.com
```

9. Customize the inventory file to identify the hosts by IP addresses.

On the **console** VM, navigate to the **/root/D0380/labs/prep-install/openshift-cluster/mycluster.com/** directory and review the **hosts-mycluster.fixme** file.

The file contains extra settings for the inventory file. Append the content of the file to the **hosts-mycluster** inventory file.

- 9.1. On the **console** VM, navigate to the **/root/D0380/labs/prep-install/openshift-cluster/mycluster.com** directory.

```
[root@console mycluster.com]# cd /root/D0380/labs/prep-install/ \
openshift-cluster/mycluster.com
```

Review the **hosts-mycluster.fixme** file, which reads as follows:

```
[masters]
master1.mycluster.com openshift_ip=10.0.1.11 openshift_public_ip=10.0.1.11
master2.mycluster.com openshift_ip=10.0.1.12 openshift_public_ip=10.0.1.12
master3.mycluster.com openshift_ip=10.0.1.13 openshift_public_ip=10.0.1.13

[nodes]
master1.mycluster.com openshift_node_labels="{'region':
'master'}" openshift_schedulable=false openshift_ip=10.0.1.11
openshift_public_ip=10.0.1.11
master2.mycluster.com openshift_node_labels="{'region':
'master'}" openshift_schedulable=false openshift_ip=10.0.1.12
openshift_public_ip=10.0.1.12
master3.mycluster.com openshift_node_labels="{'region':
'master'}" openshift_schedulable=false openshift_ip=10.0.1.13
openshift_public_ip=10.0.1.13
infra1.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.51 openshift_public_ip=10.0.1.51
infra2.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.52 openshift_public_ip=10.0.1.52
infra3.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.53 openshift_public_ip=10.0.1.53
infra4.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.54 openshift_public_ip=10.0.1.54
```

```

infra5.mycluster.com openshift_node_labels="{'region': 'infra'}"
  openshift_ip=10.0.1.55 openshift_public_ip=10.0.1.55
apps1.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.60 openshift_public_ip=10.0.1.60
apps2.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.61 openshift_public_ip=10.0.1.61
apps3.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.62 openshift_public_ip=10.0.1.62
apps4.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.63 openshift_public_ip=10.0.1.63
apps5.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.64 openshift_public_ip=10.0.1.64
apps6.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.65 openshift_public_ip=10.0.1.65
apps7.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.66 openshift_public_ip=10.0.1.66
apps8.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.67 openshift_public_ip=10.0.1.67
apps9.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.68 openshift_public_ip=10.0.1.68
apps10.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
  'local'}" openshift_ip=10.0.1.69 openshift_public_ip=10.0.1.69

```

- 9.2. Add the content of the file to the main inventory file, **hosts-mycluster**. Optionally, run the **cat** command with a redirection to append the content to the main inventory file.

```
[root@console mycluster.com]# cat hosts-mycluster.fixme >> \
/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster
```

10. From the **/root/openshift-install/openshift-cluster/mycluster.com** directory, push the inventory file to your local Git repository. Use **Ansible HA master file** as the commit message.

- 10.1. Go to the **/root/openshift-install/openshift-cluster/mycluster.com** directory.

```
[root@console mycluster.com]# cd /root/openshift-install/openshift-cluster/
mycluster.com
```

- 10.2. Use the **git add** command to update the index with the current modifications.

```
[root@console mycluster.com]# git add .
```

- 10.3. Commit your changes with the commit message **OCP HA master file**.

```
[root@console mycluster.com]# git commit -m "Ansible HA master file"
...
1 file changed, 47 insertions(+)
```

- 10.4. Push the changes to the Git repository running on the **services** VM.

```
[root@console mycluster.com]# git push
Counting objects: 9, done.
```

```
Delta compression using up to 2 threads.
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 1.03 KiB | 0 bytes/s, done.
Total 5 (delta 1), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install
  e182727..c35ff0d master -> master
```

11. Grade your work.

Run the **lab prep-install grade** script from the **workstation** VM to verify your work.

```
[student@workstation ~]$ lab prep-install grade
```

This concludes this lab.

Summary

In this chapter, you learned:

- OpenShift Container Platform provides two different installation methods, based on Ansible, for deploying an OpenShift Platform Container cluster; the quick method and the advanced method. The method chosen depends on the goals of the deployment because both methods are supported for production.
- The advanced installation method can deploy multiple servers simultaneously by using the inventory file for the playbook used to install OpenShift Container Platform. The inventory file contains a list of the systems that Ansible interacts with during the deployment.
- Red Hat recommends creating custom Ansible Playbooks for pre- and post-configuration instead of using shell scripts or manual commands.
- The OpenShift Container Platform advanced installation uses playbooks that require host names and variables to be defined in an inventory file, which supports the installation. The inventory file contains the configuration for the OpenShift Container Platform cluster.
- OpenShift Container Platform provides an integrated container registry called OpenShift Container Registry (OCR), which adds the ability to automatically provision new image repositories on demand.
- During an initial installation of a full OpenShift cluster, it is likely that the registry was deployed automatically. While it can be deployed to run as an integrated part of a full OpenShift cluster, the OpenShift Container Registry (OCR) can alternatively be installed separately as a stand-alone private container image registry.



CHAPTER 3

CONFIGURING OPENSHIFT TO USE CUSTOM CERTIFICATES

Overview	
Goal	Configure an OpenShift cluster to use custom certificates.
Objectives	<ul style="list-style-type: none">Describe supported strategies for OpenShift certificate management.Create custom certificates and configure OpenShift to use them.
Sections	<ul style="list-style-type: none">Describing an OpenShift Certificate Management Strategy (and Quiz)Configuring Custom Certificates (and Guided Exercise)
Lab	Configuring OpenShift to Use Custom Certificates

Describing an OpenShift Certificate Management Strategy

Objective

After completing this section, students should be able to describe supported strategies for OpenShift certificate management.

Certificates in OpenShift

OpenShift configures a certificate authority (CA) and the following set of certificates to enforce:

- Communication channels encryption
- A handshake protocol between components
- Authenticate users in OpenShift

For the quick installer, the certificates are self signed and use an internal CA (**openshift-signer**) that supports most use cases in a private cloud.

However, some companies have certificates from a trusted CA, and require the use of these certificates throughout the company as part of their security policy.

In OpenShift, these certificates can be used by the internal components as long as they are configured and installed by the advanced installer.

Certificates are used in the following places internally by OpenShift:

OpenShift Certificate Usage

Source (Client)	Destination (Server)	Certificate Type
Elasticsearch	Elasticsearch	Raw client certificate authentication
Etcd	Etcd	Raw client certificate authentication
Fluentd	Elasticsearch	Raw client certificate authentication
Kibana	Elasticsearch	Raw client certificate authentication
Nodes	Master API	Infrastructure component authentication via X.509 client certificate
Master	Node (kubelet)	Raw client certificate authentication
Master	Etcd	Raw client certificate authentication
system:admin user	Master API	Infrastructure component authentication via X.509 client certificate

Source (Client)	Destination (Server)	Certificate Type
Router	Master API	Service account API token
Registry	Master API	Service account API token
Registry Console	Registry	Infrastructure component authenticating via X.509 client certificate
Master controllers	Master API	Infrastructure component authenticating via X.509 client certificate
User	Master API	API token over HTTPS connection
User	Router	HTTPS connection
User	Kibana (Logging)	API token over HTTPS connection
User	Hawkular	API token over HTTPS connection
User	Registry	API token over HTTPS connection
User	Registry console	API token over HTTPS connection

Most of the heavy work is done by Ansible Playbooks provided by OpenShift, and they can be customized for the installation process.

Using a Trusted Intermediate CA

The most comprehensive approach is to provide Ansible Playbooks with a trusted intermediate CA key, and to allow the installation process to generate all certificates needed by OpenShift internals. The major benefits are:

- *Maintenance*: Ansible automatically generates the certificates for OpenShift infrastructure components, such as routers, registry, logging, and metrics.
- *Trusted source*: The certificates are based on a trusted CA that is reliably accepted by any system.
- *Renews*: Any expired certificate can be renewed by existing Ansible Playbooks provided and maintained by the OpenShift maintenance team.

To guarantee security, the trusted intermediate CA key must remain left on the OpenShift runtime environment during the following processes:

- Initial installation
- Certificate updates
- Installation of new components

Using Component-specific Certificates

You can also use a custom server certificate for each user-facing endpoint.



Warning

In order to use a component specific certificate, master cluster public facing host name and master cluster host name must be different.

A major advantage of this approach is flexibility, because you can add new trusted certificates to individual components as needed. The downside is that the internal infrastructure certificates remain self-signed, which may be perceived as bad practice by some security teams. As long as the communication channels between the OpenShift nodes are on a private network, the risks are minimal.

Certificate Storage

In an OpenShift cluster, all certificates are stored in the **/etc/origin/master** and **/etc/origin/node** directories for master nodes, and in **/etc/origin/node** for the remaining nodes:

```
[root@masterX ~]# tree /etc/origin/master/
/etc/origin/master/
├── admin.crt
├── admin.key
├── admin.kubeconfig
├── ca-bundle.crt ①
├── ca.crt
├── ca.key
└── ca.serial.txt
├── etcd.server.crt ②
├── etcd.server.key
├── htpasswd
└── master-config.yaml
├── master.etcd-ca.crt ③
├── master.etcd-client.crt ④
├── master.etcd-client.csr
└── master.etcd-client.key
├── master.kubelet-client.crt ⑤
├── master.kubelet-client.key
├── master.proxy-client.crt
├── master.proxy-client.key
├── master.server.crt
└── master.server.key
└── named_certificates
├── openshift-master.crt ⑥
├── openshift-master.key
└── openshift-master.kubeconfig
└── policy.json
├── registry.crt ⑦
├── registry.key
└── scheduler.json
├── serviceaccounts.private.key
└── serviceaccounts.public.key
├── service-signer.crt
└── service-signer.key
└── session-secrets.yaml
```

- ① Trusted CA certificate
- ② Etcd server certificate
- ③ Master to Etcd client certificate
- ④ Client certificate for Etcd access
- ⑤ Master to node certificate
- ⑥ OpenShift service authentication certificates
- ⑦ OpenShift registry authentication certificates

All these certificates are configured in the `/etc/origin/master/master-config.yaml` file.

```
[root@masterX ~]# tree /etc/origin/node
/etc/origin/node
├── ca.crt
├── node-config.yaml
├── node-dnsmasq.conf
├── resolv.conf
└── server.crt①
    └── server.key
        ├── system:node:master1.lab.example.com.crt②
        ├── system:node:master1.lab.example.com.key
        └── system:node:master1.lab.example.com.kubeconfig
```

- ① Master to node certificate
- ② Node to Server authentication certificate

Most node certificates are configured in the `/etc/origin/node/node-config.yaml` file.



Important

You can update these configuration files manually, but because of the number of changes needed, Red Hat strongly recommends that you update them using the advanced installer.



References

Further information is available in the Configuring Custom Certificates chapter of the *Installation and Configuration Guide* for OpenShift Container Platform; at
<https://access.redhat.com/documentation/en-US/index.html>

Quiz: Describing OpenShift Certificate Management Strategies

Choose the correct answers to the following questions:

1. For which three reasons do OpenShift Container Platform internal services use certificates? (Choose three.)
 - a. To encrypt channel communication
 - b. To authenticate users
 - c. To encrypt data stored in a persistent volume
 - d. A handshake protocol between components

2. Which four certificate types are used by OpenShift Container Platform internal services? (Choose four.)
 - a. Raw client certificate authentication
 - b. Service account API token
 - c. API token over HTTPS connection
 - d. MD5 authentication tokens
 - e. X.509 client certificates
 - f. Base64 authentication tokens

3. Which of the following is an advantage of using OpenShift Container Platform component-specific certificates?
 - a. It provides the flexibility to use a certificate on each component, without changing the whole certificate chain.
 - b. You can implement a single sign-on approach on all applications running on OpenShift Container Platform.
 - c. You can focus on the certification management for the each component-specific certificate.

4. What are three advantages of using OpenShift Container Platform trusted intermediate certificate authority (Choose three?)
 - a. You can use a certificate on each application, without changing the whole certificate chain.
 - b. Ansible automatically generates the certificates for OpenShift Container Platform infrastructure components, such as routers, registry, logging, and metrics.
 - c. A trusted CA provides the certificate and it is reliably accepted by any system.
 - d. Ansible Playbooks provided by OpenShift Container Platform automatically update the certificates whenever a change is needed.

Solution

Choose the correct answers to the following questions:

1. For which three reasons do OpenShift Container Platform internal services use certificates? (Choose three.)
 - a. To encrypt channel communication
 - b. To authenticate users
 - c. To encrypt data stored in a persistent volume
 - d. A handshake protocol between components
2. Which four certificate types are used by OpenShift Container Platform internal services? (Choose four.)
 - a. Raw client certificate authentication
 - b. Service account API token
 - c. API token over HTTPS connection
 - d. MD5 authentication tokens
 - e. X.509 client certificates
 - f. Base64 authentication tokens
3. Which of the following is an advantage of using OpenShift Container Platform component-specific certificates?
 - a. It provides the flexibility to use a certificate on each component, without changing the whole certificate chain.
 - b. You can implement a single sign-on approach on all applications running on OpenShift Container Platform.
 - c. You can focus on the certification management for the each component-specific certificate.
4. What are three advantages of using OpenShift Container Platform trusted intermediate certificate authority (Choose three?)
 - a. You can use a certificate on each application, without changing the whole certificate chain.
 - b. Ansible automatically generates the certificates for OpenShift Container Platform infrastructure components, such as routers, registry, logging, and metrics.
 - c. A trusted CA provides the certificate and it is reliably accepted by any system.
 - d. Ansible Playbooks provided by OpenShift Container Platform automatically update the certificates whenever a change is needed.

Configuring Custom Certificates

Objective

Create custom certificates and configure OpenShift to use them.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Using a Trusted Intermediate Certificate Authority (CA)

To work with a trusted intermediate CA, OpenShift requires both the certificate and the key. In the Ansible inventory file used by the OpenShift advanced installer, a variable in the **[OSEv3:vars]** group called **openshift_master_ca_certificate** must be used to refer to both the certificate and the key. The certificate and key files must be stored in a directory accessible to Ansible. Fortunately, the installation process automatically generates all certificates as part of the installation.

```
[OSEv3:vars]
...
openshift_master_ca_certificate={'certfile': '/home/student/ca.crt', 'keyfile': '/home/student/ca.key'}
```

The benefits of this approach are as follows:

- *Centralized certification generation:* A trusted CA signs all certificates in the cluster and they are generated for you at configuration time, including router wildcard, registry, logging, metrics, and any integrated services.
- *Automatic updates:* Whenever a certificate expires, Ansible Playbooks are available to automatically renew certificates.



Important

You can use the OpenShift Master API to update certificates, but this approach is error prone. Red Hat recommends that you install the certificates during the initial installation to minimize risk.

Using a Component-specific Certificate

Each OpenShift component may have its own certificate, but each component must be configured as part of the Ansible inventory file, in the **[OSEv3:vars]** group.

OpenShift Master API Certificate

To facilitate trusted connections with external users of OpenShift, a named certificate can be provisioned that matches the domain name provided in the **openshift_master_cluster_public_hostname** variable. This certificate must be placed in a directory accessible to Ansible, and added to the Ansible inventory file, as follows:

```
openshift_master_named_certificates=[  
    { "certfile": "/path/to/console.mycluster.com.crt",  
      "keyfile": "/path/to/console.mycluster.com.key",  
      "names": ["console.mycluster.com"],  
      "cafefile": "/path/to/console.mycluster.com.ca.crt"  
    }  
]
```

Wildcard Router Certificate

A default wildcard certificate may be used with the OpenShift router, which provides a convenient way to encrypt communication without the need to create a certificate for each route. To configure a default wildcard certificate, provision a certificate that is valid for the application domain. After it has been provisioned, place the certificate, key, and CA certificate files in a directory accessible to Ansible, and add the following lines to the Ansible inventory file in the **[OSEv3:vars]** group.

```
openshift_hosted_router_certificate={  
    "certfile": "/path/to/apps.mycluster.com.crt",  
    "keyfile": "/path/to/apps.mycluster.com.key",  
    "cafefile": "/path/to/apps.mycluster.com.ca.crt"}
```

Image Registry Certificate

A re-encrypt route exposes the registry to allow external systems and users to log in to the registry to pull and push images. To configure the route, add the following lines to the Ansible inventory file, pointing to the certificate files.

```
openshift_hosted_registry_routehost=registry.mycluster.com  
openshift_hosted_registry_routecertificates={  
    "certfile": "/path/to/registry.mycluster.com.crt",  
    "keyfile": "/path/to/registry.mycluster.com.key",  
    "cafefile": "/path/to/registry.mycluster.ca.crt"}
```

Managing Certificates

OpenShift provides an Ansible Playbook to manage certificates and simplify the update process. The playbooks are available at **/usr/share/ansible/openshift-ansible/playbooks/certificate_expiry/** and they use the Ansible inventory file used by the advanced installer to inspect all hosts.

```
# ansible-playbook -v -i inventory-file \  
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster/playbook-file
```

For example, the **easy-mode.yml** playbook identifies expired certificates and exports the list to the **/tmp** directory.

Use the **redeploy-openshift-ca.yml** file with the Ansible inventory file to update an expired intermediate CA.

Finally, for the component specific certificate scenario, the **redeploy-*-certificates.yml** can be used.



References

Further information is available in the Redeploying Certificates chapter of the *OpenShift Installation and Configuration Guide* for OpenShift Container Platform 3.6; at
<https://access.redhat.com/documentation/en-US/index.html>

Guided Exercise: Configuring OpenShift to use an Intermediate Certificate Authority (CA)

In this exercise, you will configure a custom certificate authority in OpenShift to be used by all applications running on OpenShift.

Outcomes

You should be able to configure an Ansible inventory file used by OpenShift advanced installer to use an intermediate CA.



Note

The **mycluster.com** domain, used in examples and exercises through chapter 3, is fictitious. Starting in chapter 4, and for the remainder of this course, the working cluster uses the reserved training domain name **lab.example.com**.

Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab cert-files setup** command to download the required files for this exercise.

```
[student@workstation ~]$ lab cert-files setup
```

Steps

1. Ensure that the Ansible inventory file is up-to-date on the **console** VM.

1.1. Access the **console** VM.

On the **workstation** VM, open a new terminal and use the **ssh** command to log in to **console** as the **root** user.

```
[student@workstation ~]$ ssh root@console
```

1.2. Ensure that the inventory file is present in the **/root/openshift-install/openshift-cluster/mycluster.com** directory on the **console** VM.

Ensure that the **hosts-mycluster** inventory file, created in the previous exercise, exists in the **/root/openshift-install/openshift-cluster/mycluster.com** directory on the **console** VM.

```
[root@console ~]# cd /root/openshift-install/openshift-cluster/mycluster.com
[root@console mycluster.com]# ls
hosts-mycluster
```

1.3. Use a text editor to inspect the file and compare it to the following output:

```
# hosts file for mycluster.com
[OSEv3:children]
masters
```

```
etcd
nodes
nfs
lb

[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
deployment_type=openshift-enterprise
openshift_release=v3.6
osm_cluster_network_cidr=10.128.0.0/14
openshift_portal_net=172.30.0.0/16
openshift_master_api_port=443
openshift_master_console_port=8443
openshift_master_cluster_public_hostname=console.mycluster.com
openshift_master_cluster_hostname=console.mycluster.com
openshift_master_default_subdomain=apps.mycluster.com
openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/
origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$Y9pwjRCD$iqLLsX/
vhUljsTxiYq9FY.',
'developer': '$apr1$AH9Loha.$GJNmEjjKjt/8LPET01v751'}
openshift_master_cluster_method=native

openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
openshift_docker_additional_registries=registry.mycluster.com:5000
openshift_docker_insecure_registries=registry.mycluster.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io

openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'

oreg_url=registry.lab.example.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

openshift_set_node_ip=True

openshift_disable_check=disk_availability,docker_storage,memory_availability, \
docker_image_availability
ansible_ssh_user=root
openshift_cockpit_deployer_prefix='openshift3/'

[nfs]
nfs.mycluster.com

[lb]
lb.mycluster.com containerized=false

[etcd]
etcd[1:5].mycluster.com

[masters]
master1.mycluster.com openshift_ip=10.0.1.11 openshift_public_ip=10.0.1.11
master2.mycluster.com openshift_ip=10.0.1.12 openshift_public_ip=10.0.1.12
master3.mycluster.com openshift_ip=10.0.1.13 openshift_public_ip=10.0.1.13

[nodes]
```

```

master1.mycluster.com openshift_node_labels="{'region':
    'master'}" openshift_schedulable=false openshift_ip=10.0.1.11
openshift_public_ip=10.0.1.11
master2.mycluster.com openshift_node_labels="{'region':
    'master'}" openshift_schedulable=false openshift_ip=10.0.1.12
openshift_public_ip=10.0.1.12
master3.mycluster.com openshift_node_labels="{'region':
    'master'}" openshift_schedulable=false openshift_ip=10.0.1.13
openshift_public_ip=10.0.1.13
infra1.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.51 openshift_public_ip=10.0.1.51
infra2.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.52 openshift_public_ip=10.0.1.52
infra3.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.53 openshift_public_ip=10.0.1.53
infra4.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.54 openshift_public_ip=10.0.1.54
infra5.mycluster.com openshift_node_labels="{'region': 'infra'}"
openshift_ip=10.0.1.55 openshift_public_ip=10.0.1.55
apps1.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.60 openshift_public_ip=10.0.1.60
apps2.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.61 openshift_public_ip=10.0.1.61
apps3.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.62 openshift_public_ip=10.0.1.62
apps4.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.63 openshift_public_ip=10.0.1.63
apps5.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.64 openshift_public_ip=10.0.1.64
apps6.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.65 openshift_public_ip=10.0.1.65
apps7.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.66 openshift_public_ip=10.0.1.66
apps8.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.67 openshift_public_ip=10.0.1.67
apps9.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.68 openshift_public_ip=10.0.1.68
apps10.mycluster.com openshift_node_labels="{'region': 'primary', 'zone':
    'local'}" openshift_ip=10.0.1.69 openshift_public_ip=10.0.1.69

```

Update the file if any differences exist.



Note

You can also use the `/root/D0380/labs/cert-files/openshift-cluster/mycluster.com/hosts-mycluster` inventory file as the base file.

- 1.4. Run the `git pull` command to ensure that the latest changes are present in the inventory file.

```
[root@console mycluster.com]# git pull
Already up-to-date.
```

2. Inspect the script that creates the intermediate certificate.

- 2.1. On the **console** VM, inspect the **cert.sh** script located in the **/root/D0380/labs/cert-files** directory.

The script creates the private key, the certificate signing request, and the certificate.

```
openssl genrsa -out /root/openshift-install/openshift-cluster/mycluster.com/
rootCA.key 2048 ①
openssl req -new -key /root/openshift-install/openshift-cluster/mycluster.com/
rootCA.key \
-out /root/openshift-install/openshift-cluster/mycluster.com/rootCA.csr \
-subj "/C=US/ST=NC/L=Raleigh/O=Red Hat Training/OU=IT/CN=mycluster.com" ②
openssl x509 -req -days 366 \
-in /root/openshift-install/openshift-cluster/mycluster.com/rootCA.csr \
-signkey /root/openshift-install/openshift-cluster/mycluster.com/rootCA.key \
-out /root/openshift-install/openshift-cluster/mycluster.com/rootCA.crt ③
```

- ① Generates the root certificate for the installation
- ② Generates the certificate signing request (CSR)
- ③ Generates the certificate using the key and the CSR

- 2.2. Run the script to generate the certificates.

On the **console** VM terminal window, run the following commands:

```
[root@console mycluster.com]# cd /root/D0380/labs/cert-files
[root@console cert-files]# ./cert.sh
```

- 2.3. List the files generated by the script.

Inspect the contents of the directory **/root/openshift-install/openshift-cluster/mycluster.com**.

```
[root@console cert-files]# ls \
/root/openshift-install/openshift-cluster/mycluster.com
hosts-mycluster rootCA.crt rootCA.csr rootCA.key
```

3. Add the OpenShift certificate configuration to the Ansible inventory file.



Note

To avoid typing errors, you can use the **/root/D0380/labs/cert-files/cert-config** configuration file.

Update the **/root/openshift-install/openshift-cluster/mycluster.com/hosts-mycluster** file to include the following line in the **[OSEv3:vars]** group.

```
openshift_master_ca_certificate={'certfile':'/root/openshift-install/openshift-
cluster/mycluster.com/rootCA.crt','keyfile':'/root/openshift-install/openshift-
cluster/mycluster.com/rootCA.key'}
```



Note

The previous content must be typed in a single line.

This file should now appear as follows:

```
# hosts file for mycluster.com
[OSEv3:children]
masters
etcd
nodes
nfs
lb

[OSEv3:vars]
...
oreg_url=registry.lab.example.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

openshift_master_ca_certificate={'certfile': '/root/openshift-install/openshift-cluster/mycluster.com/rootCA.crt', 'keyfile': '/root/openshift-install/openshift-cluster/mycluster.com/rootCA.key'}

[nfs]
nfs.mycluster.com

...
```

4. From **workstation**, open a new terminal and run the **lab cert-files grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab cert-files grade
...
Comparing Entries in [OSEv3:vars]

· Checking openshift_deployment_type..... PASS
· Checking deployment_type..... PASS
· Checking openshift_release..... PASS
· Checking osm_cluster_network_cidr..... PASS
· Checking openshift_portal_net..... PASS
...
```

5. Push the changes to Git.

- 5.1. From the **console** VM, add the Ansible inventory file to the Git repository.

From the **console** VM, go to the **/root/openshift-install** directory and add all the contents from the **openshift-cluster** directory to the Git repository.

```
[root@console cert-files]# cd /root/openshift-install
[root@console openshift-install]# git add openshift-cluster/mycluster.com/*
```

- 5.2. Commit the Ansible inventory file, the private key, and the certificate signing request to the Git repository with the comment **Configuring the intermediate certificate**.

```
[root@console openshift-install]# git commit -m \
'Configuring the intermediate certificate'
```

- 5.3. Push the **openshift-cluster** project directory to the Git repository.

```
[root@console openshift-install]# git push
...
Counting objects: 12, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (7/7), done.
Writing objects: 100% (8/8), 4.44 KiB | 0 bytes/s, done.
Total 8 (delta 1), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install
a61e7b1..9d01a61 master -> master
```

This concludes this guided exercise.

Lab: Configuring OpenShift to Use Custom Certificates

Outcomes

You should be able to configure an Ansible inventory file used by the OpenShift advanced installer to use an intermediate CA.

Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab cert-review setup** command to download the required files for this exercise.

```
[student@workstation ~]$ lab cert-review setup
```

Steps

1. Inspect the Ansible inventory file created in the previous chapter. Pull it from the **services.lab.example.com** Git repository on the **console** VM and inspect its content.
2. Inspect and run the **/root/D0380/labs/cert-review/cert-review.sh** script responsible for creating the intermediate certificate.
3. Add the OpenShift certificate configuration to the **/root/openshift-install/lab.example.com/hosts-ucf** Ansible inventory file.
4. Push the changes to Git.
5. Run the **lab cert-review grade** script on the **workstation** VM to grade your work.

```
[student@workstation ~]$ lab cert-review grade
```

This concludes this lab.

Solution

Outcomes

You should be able to configure an Ansible inventory file used by the OpenShift advanced installer to use an intermediate CA.

Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab cert-review setup** command to download the required files for this exercise.

```
[student@workstation ~]$ lab cert-review setup
```

Steps

1. Inspect the Ansible inventory file created in the previous chapter. Pull it from the **services.lab.example.com** Git repository on the **console** VM and inspect its content.

- 1.1. From the **workstation** VM, log in to the **console** VM as the **root** user.

```
[student@workstation ~]$ ssh root@console
```

- 1.2. Navigate to the Git local repository and run the **git pull** command to ensure that the latest changes are present in the inventory file.

```
[root@console ~]$ cd openshift-install/lab.example.com
[root@console lab.example.com]# git pull
Already up-to-date.
```

2. Inspect and run the **/root/D0380/labs/cert-review/cert-review.sh** script responsible for creating the intermediate certificate.

- 2.1. On the **console** VM, inspect the **/root/D0380/labs/cert-review/cert-review.sh** script.

The script creates the private key, the certificate signing request, and the certificate.

```
openssl genrsa -out /root/openshift-install/lab.example.com/rootCA.key 2048①
openssl req -new -key /root/openshift-install/lab.example.com/rootCA.key \
-out /root/openshift-install/lab.example.com/rootCA.csr \
-subj "/C=US/ST=NC/L=Raleigh/O=Red Hat Training/OU=IT/CN=lab.example.com"②
openssl x509 -req -days 366 -in /root/openshift-install/lab.example.com/
rootCA.csr \
-signkey /root/openshift-install/lab.example.com/rootCA.key \
-out /root/openshift-install/lab.example.com/rootCA.crt ③
```

① Generates the root certificate for the installation

② Generates the certificate signing request (CSR)

③ Generates the certificate using the key and the CSR

- 2.2. Run the script to generate the certificates.

On the **console** VM, run the following commands:

```
[root@console lab.example.com]# cd /root/D0380/labs/cert-review
[root@console cert-review]# ./cert-review.sh
```

2.3. List the files generated by the script.

List the contents of the **/root/openshift-install/lab.example.com** directory.

```
[root@console cert-review]# ls /root/openshift-install/lab.example.com
hosts-ucf README.md rootCA.crt rootCA.csr rootCA.key
```

3. Add the OpenShift certificate configuration to the **/root/openshift-install/lab.example.com/hosts-ucf** Ansible inventory file.

Update the **/root/openshift-install/lab.example.com/hosts-ucf** file to include the following line in the **[OSEv3:vars]** group.

Solution

The entry is a single line.

```
openshift_master_ca_certificate={'certfile': '/root/openshift-install/
lab.example.com/rootCA.crt', 'keyfile': '/root/openshift-install/lab.example.com/
rootCA.key'}
```

4. Push the changes to Git.

4.1. Add the updated Ansible inventory file to the Git index.

Go to the **/root/openshift-install** directory and add all the updates to the Git index.

```
[root@console ~]# cd /root/openshift-install
[root@console openshift-install]# git add lab.example.com/*
```

4.2. Commit the Ansible inventory file, the private key, and the certificate signing request to the Git repository with the comment **Configuring the intermediate certificate**.

```
[root@console openshift-install]# git commit -m \
'Configuring the intermediate certificate'
```

4.3. Push the **openshift-cluster** project directory to the Git repository.

```
[root@console openshift-install]# git push
...
Counting objects: 10, done.
Delta compression using up to 2 threads.
```

```
Compressing objects: 100% (7/7), done.  
Writing objects: 100% (7/7), 3.47 KiB | 0 bytes/s, done.  
Total 7 (delta 1), reused 0 (delta 0)  
To http://services.lab.example.com/openshift-install  
9d01a61..37260b7 master -> master
```

5. Run the **lab cert-review grade** script on the **workstation** VM to grade your work.

```
[student@workstation ~]$ lab cert-review grade
```

This concludes this lab.

Summary

In this chapter, you learned that:

- OpenShift Container Platform can be used to manage security certificates for multiple pods in a centralized way.
- OpenShift's internal certificate management model guarantees encryption for all components, from node communication to Etcd management.
- Certificates can be managed on a service basis approach.
- OpenShift automatically updates certificates using Ansible Playbooks.
- OpenShift provides playbooks to manage certificates installed on OpenShift, including identifying expired certificates, updating an intermediate CA and its associated certificates, as well as updating individual certificates.



CHAPTER 4

BUILDING A HIGHLY AVAILABLE CLUSTER

Overview	
Goal	Build a highly available OpenShift cluster using the advanced installation method.
Objectives	<ul style="list-style-type: none">Provision servers that will make up the highly available cluster.Execute the required tasks before using the advanced installation method.Use the advanced installation method to build the cluster.
Sections	<ul style="list-style-type: none">Provisioning Servers (and Guided Exercise)Executing Pre-installation Tasks (and Guided Exercise)Running the Advanced Installation Method (and Guided Exercise)
Lab	Building a Highly Available Cluster

Provisioning Servers

Objective

After completing this section, students should be able to provision servers that make up a *highly available (HA)* cluster.

Deployment Scenarios for OpenShift Container Platform?

With the increase in performance of virtualization technologies and abstraction layers, performance in virtualized environments closely matches the performance of physical hardware. Three options are typically available for deploying OpenShift Container Platform: bare metal, virtualization solutions, and cloud computing solutions. All these options have their benefits, the decisive factors being deployment and maintenance costs, administrator training, and ROI.

Deploying on Bare Metal

OpenShift Container Platform can be deployed to a bare metal environment. This deployment generally offers increased performance because the latency of disks and networks is minimized. In this deployment, a physical server fulfills one role in the cluster, such as an applications node, or a master. When deploying on bare metal, hardware failure leads to availability failure for the particular node. To circumvent that, advanced deployment of disks through RAID, or network via bonding, is often required. Doing so leads to an increase in cost and maintenance.

Deploying on Virtualized Environments

Virtualized environments, such as Red Hat Virtualization or VMware ESX, are similar to bare metal deployments insofar as performance and complexity of administration are concerned. A virtualized platform does not suffer from the same caveats as bare metal environments do; for example, virtual instances can be migrated upon hardware failure. Hardware failure does not impinge on the cluster availability the same way bare metal environments do.

Virtualized environments often offer an extra layer of abstraction, such as APIs, which can be used to programmatically deploy the required instances. Virtual machines can be deployed via a web console or a command-line interface, and offer features such as backups, network monitoring and administration, and reports. Virtualized environments often offer tooling, automation tools, or integration with tier products.

For example, OpenShift Container Platform ships with CloudForms for operational management. Red Hat Virtualization can be integrated with Ansible, itself used for the deployment of OpenShift Container Platform. When you choose virtualized environments, costs tend to be more predictable, because vendors base their costs on the number of sockets.

Moreover, enterprises can benefit from support of their virtualization software, which includes security audits, patching, and access to continuous support.

Cloud Environments

Cloud environments, such as Amazon EC2, Red Hat OpenStack Platform, and Google Compute Engine, are designed to run distributed workloads and are fully supported by OpenShift Container Platform. These environments scale as needed, and the cost is based on the consumption of resources. These environments focus on automation and availability of computing resources.

Cloud environments that are deployed internally (referred as *private clouds*), have a steeper learning curve than bare-metal environments. On the contrary, cloud environments served online by a third party (referred as *public clouds*), often cost less to operate, because there are no hardware-associated costs, nor licenses costs. With the public cloud model, companies pay for the resources they consume and have access to real-time metrics to precisely measure such resources. Therefore, companies should factor in the cost associated with hardware, maintenance, training, and infrastructure design.

Deployment Scenarios

Administrators can deploy and configure their environment in various fashions:

- Single master and multiple nodes
- Single master, multiple Etcd servers, and multiple nodes
- Multiple masters with OpenShift native HA, using HAProxy as the load balancer
- Stand-alone registry server with a single master or multiple master nodes

You can separate nodes by function. For example, you can dedicate a set of servers for infrastructure nodes, to run Red Hat Container Registry and GlusterFS services. You can also dedicate a set of servers for application nodes, to run the user applications.



Note

The classroom environment uses a set of master, infrastructure, and application nodes. This environment constrains the allocation of resources due to hardware limitations.



Note

The hardware and storage requirements presented below differ between minor releases of OpenShift Container Platform. Therefore, the values suggested might may be different in the future.

For more information, consult the *Scaling and Performance Guide* listed in the references.

Hardware Requirements

The system requirements vary per host type. The following table lists the system requirements for master nodes, nodes, and Etcd servers.

Node Type	Hardware Requirements
Masters	<ul style="list-style-type: none">• A physical or a virtual machine. For example, a virtual machine running in a public or private cloud.• Red Hat Enterprise Linux version 7.3 or 7.4, for AMD64 and Intel 64, with a <i>minimal</i> installation and latest packages from the Extras channel, or RHEL Atomic Host 7.3.6 or later.• Two vCPUs, with a minimum of 16 GB of RAM.

Node Type	Hardware Requirements
	<ul style="list-style-type: none"> 40 GB of disk space for the /var/ mount point and 1GB of disk space for the /usr/local/bin mount point and the file system containing the system's /tmp partition. In a highly available OpenShift Container Platform cluster with external Etcd instances, an additional 1CPU and 1.5 GB of memory for each 1000 pods are required.
Nodes	<ul style="list-style-type: none"> A physical or a virtual machine, for example, an instance running on a public or private cloud. Red Hat Enterprise Linux version 7.3 or 7.4 for AMD64 and Intel 64, with a <i>minimal</i> installation and latest packages from the Extras channel, or RHEL Atomic Host 7.3.6 or later. The NetworkManager service must run be running the version 1.0 or later. One vCPU, with a minimum of 1GB of RAM. 15 GB of disk space for the /var/ mount point and 1GB of disk space for the /usr/local/bin mount point and the file system containing the system's /tmp partition. An additional unallocated 15 GB of disk space is required. The space is used for the Docker service storage back end.
Etcd	<ul style="list-style-type: none"> A minimum of 20 GB of disk space for Etcd data. An additional 15 GB of unallocated disk space is required. The space is used for the Docker storage back end.

Storage Requirements

If you choose to configure containerized GlusterFS persistent storage for the cluster, or if you choose to configure OpenShift Container Registry with Gluster, you must consider the following prerequisites.

Node Type	Storage Requirements
Docker Service	<ul style="list-style-type: none"> For 20 - 100 containers, Red Hat recommends 20 - 50 GB of disk space. The Docker service stores the data in the /var/lib/docker directory.
GlusterFS Storage Nodes	<ul style="list-style-type: none"> A minimum of three storage nodes are required. Each storage node must have at least one block device with at least 100 GB available.
Container Registry with GlusterFS	<ul style="list-style-type: none"> A minimum of three storage nodes are required. Each storage node must have at least one block device with at least 10 GB available.



Note

Gluster CNS requirements are covered in a later chapter.



Note

By default, masters and nodes use all available cores in the system on which they run. You can set the number of core you want to allocate to OpenShift Container Platform by setting the **GOMAXPROCS** environment variable. For example, the following setting limits the number of cores to one:

```
export GOMAXPROCS=1
```

General Requirements

You must configure the following services before installing OpenShift Container Platform:

- Passwordless SSH must be configured between all servers without strict host checking from the masters. This is done by exchanging SSH keys and configuring the **StrictHostKeyChecking** variable in the **.ssh/config** configuration file:

```
Host *
  StrictHostKeyChecking no
```

- A Red Hat account is required with an active OpenShift Container Platform subscription. The subscription includes the required channels.
- The installation playbook is available by installing the *atomic-openshift-utils* package on the server that runs the deployment. The package creates various playbooks in the **/usr/share/ansible/openshift-ansible/playbooks/** directory.
- Security-Enhanced Linux (SELinux) must be enabled on all servers before installing OpenShift Container Platform. The **SELINUXTYPE** variable must be set to **targeted** in the **/etc/selinux/config** file:

```
SELINUXTYPE=targeted
```

- Network Time Protocol (NTP) must be configured to prevent drift between masters and nodes in the cluster. To enable NTP services, set **openshift_clock_enabled** to **true** in the Ansible inventory file during the installation.
- A shared network must exist between the master and node hosts. If cluster administrators plan to configure multiple masters for high availability using the advanced installation method, a virtual IP address must be configured during the installation process. The IP must be routable between all of the nodes. If the nodes use a fully qualified domain name, those names should resolve on all nodes.

DNS Configuration

OpenShift Container Platform requires a fully functional DNS server in the environment. Ideally, you should use a separate host running a dedicated DNS server that provides name resolution to hosts and containers running in the cluster. The following list describes the process for name resolution and the requirements for the DNS infrastructure.

The services that manage OpenShift Container Platform run inside containers, and use the following process for name resolution:

1. Containers retrieve their DNS configuration for the `/etc/resolv.conf` file from the host that they run on.
2. OpenShift Container Platform inserts one DNS value into the pods. The value is defined in the `/etc/origin/node/node-config.yaml` file, by the `dnsIP` parameter. This value is set by default to the address of the host node.
3. If the `dnsIP` parameter is omitted from the `node-config.yaml` file, the value defaults to the Kubernetes service IP address, which is the first DNS name server defined in the pod's `/etc/resolv.conf` file.



Note

As of OpenShift Container Platform 3.2, `dnsmasq` is automatically configured on all masters and nodes. The pods use the nodes as their DNS server, and the nodes forward the requests to the nodes.

By default, the `dnsmasq` service is configured on the nodes to listen on port 53. This means the nodes cannot run any other type of DNS application.

You must ensure that the **NetworkManager** dispatch script automatically configures DNS based on the DHCP configuration, if hosts retrieve their IP address via a DHCP server. You can optionally add a value to the `dnsIP` directive in the `node-config.yaml` configuration file to prepend the pod's `resolv.conf` file. The second name server is then defined by the host's first name server. By default, this becomes the IP address of the node host.

You must configure the network interface to be static, and define the DNS name server read by the **NetworkManager** service if the hosts do not retrieve their IP address from a DHCP server,



Important

Unless needed, do not set the `openshift_dns_ip` option during the advanced installation of OpenShift Container Platform using Ansible, because this option overrides the default IP address set by the `dnsIP` directive.

The installer automatically configures each node to use `dnsmasq` and forward requests to the external DNS provider.

To ensure that the host names can be resolved by the DNS server, the `/etc/resolv.conf` file must read as follows:

```
# Generated by NetworkManager
```

```
search example.com
nameserver 172.25.250.10
# nameserver updated by /etc/NetworkManager/dispatcher.d/99-origin-dns.sh
```

Run the **dig** command against a node using the DNS server defined to ensure that DNS resolution works:

```
[root@demo ~]# dig master1.lab.example.com @172.25.250.10
172.25.250.11
```

Disabling **dnsmasq**

If needed, you can disable the **dnsmasq** service. This can be the case if the environment uses a different service than **NetworkManager**. To do so, the **openshift_use_dnsmasq** value needs to be set to **False** in the Ansible Playbook.



Important

By disabling the **dnsmasq** service, some containers are not properly moved to the next name server when the first issues a **SERVFAIL**.

Configuring a DNS Wildcard

Administrators can configure a DNS wildcard, which is used by the router. A wildcard for a DNS zone must resolve to the IP address of the OpenShift Container Platform router.

For example, to create a wildcard DNS entry for the **cloudapps** subdomain that points to the public IP address of the host where the router is deployed, define the following **A** record:

```
*.cloudapps.example.com. 300 IN A 192.168.133.2
```



Important

You must ensure that the DNS server set in the **/etc/resolv.conf** file on each node host which has the wildcard entry is not listed as a name server, or that the wildcard domain is not listed in the search list. Otherwise, containers managed by OpenShift Container Platform may fail to resolve host names properly.

Pre-installation Steps

Before running the installation, there are various ways to test the environment. The following lists some verification steps.

- From the controller node, ensure that passwordless SSH is configured. By doing so, the **ssh** command can be used to connect remotely to the servers without typing the password.

```
[root@demo ~]# ssh master1
```

- Run the **dig** command against each node in the environment to ensure that DNS resolution works correctly.

```
[root@demo ~]# dig master1 @172.25.250.10
```

- Run the Ansible **ping** ad hoc module against the Ansible inventory file that defines all the servers used for the deployment.

```
[root@demo ~]# ansible -i hosts-ucf OSEv3 -m ping
```

References

Further information about sizing guidelines is available in the *Scaling and Performance Guide for OpenShift Container Platform* at

 | https://access.redhat.com/documentation/en-us/openshift_container_platform/

Further information is available in the Installing a Cluster section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at

 | <https://access.redhat.com/documentation/en-US/index.html>

Guided Exercise: Provisioning Servers

In this guided exercise, you will prepare your environment for the installation of the OpenShift cluster by reviewing resource requirements. You will run an Ansible ad hoc command that ensures connectivity between the cluster VMs.

Outcomes

You should be able to:

- Review the DNS requirements for the cluster installation.
- Review the network configuration for an infrastructure node.
- Run an Ansible ad hoc command to ensure connectivity to the cluster nodes.

Before you begin

Log in to **workstation** as **student** using **student** as the password.

Run the **lab provision-servers setup** command. The script ensures that the required OpenShift utility packages are installed on the **console** server. The script also installs the required files for this exercise.

```
[student@workstation ~]$ lab provision-servers setup
```

Steps

1. Review the following diagram, which describes the environment, comprised of the following VMs:
 - The **console** VM, which runs the OpenShift web console, a load balancer, and an NFS server.
 - The **infrastructure** VMs are load balanced by the **console** VM. The **infrastructure** VM runs the router and registry services.
 - The **master** VMs are load balanced by the **console** server. The **master** VMs run the master APIs, the replication controllers, the scheduler, and the data stores.
 - The **application** VMs run the pods of user applications.
 - The **services** VM runs the Red Hat Container Registry and stores the certificates infrastructure.
 - All nodes contain a DNS record in the **lab.example.com** subdomain. Applications are assigned to DNS names in the **apps.lab.example.com** subdomain.



Note

The classroom environment, which uses 32 GB of memory, is suited for a proof-of-concept environment. However, it is not suited for a production deployment, as it does not meet the minimum requirements.

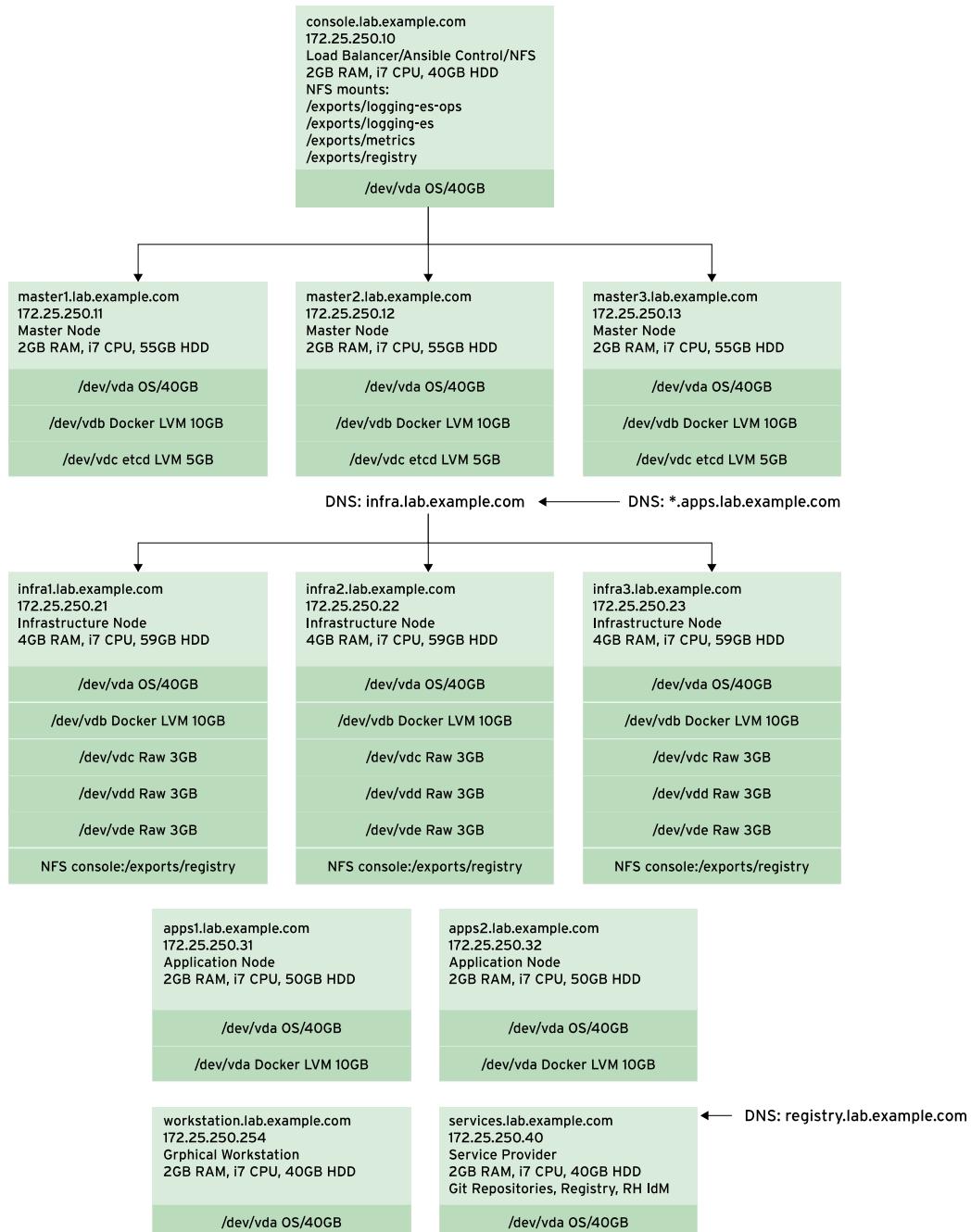


Figure 4.1: The classroom environment

2. Inspect the Ansible installation playbook on the **console** VM.

2.1. Log in to **console** as the **root** user.

On the **workstation** server, open a terminal and use the **ssh** command to connect to the **console** VM.

```
[student@workstation ~]$ ssh root@console  
[root@console ~]#
```

2.2. Ensure that the *atomic-openshift-utils* package is installed.

```
[root@console ~]# rpm -qa atomic-openshift-utils  
atomic-openshift-utils-3.6.173.0.48-1.git.0.1609d30.el7.noarch
```

2.3. List the playbooks installed by the *atomic-openshift-utils* package.

The advanced installation playbook is located at **/usr/share/ansible/openshift-ansible/playbooks/byo/**.

```
[root@console ~]# ls -al /usr/share/ansible/openshift-ansible/playbooks/byo  
total 28  
drwxr-xr-x. 10 root root 4096 Aug 30 17:53 .  
drwxr-xr-x.  9 root root 141 Aug 30 17:53 ..  
-rw-r--r--.  1 root root  44 Aug  4 07:46 config.yml  
drwxr-xr-x.  2 root root  45 Aug 30 17:53 openshift-cfme  
drwxr-xr-x.  3 root root 120 Aug 30 17:53 openshift-checks  
drwxr-xr-x.  3 root root 4096 Aug 30 17:53 openshift-cluster  
drwxr-xr-x.  2 root root  94 Aug 30 17:53 openshift-etcd  
-rw-r--r--.  1 root root 448 Aug  4 07:46 openshift_facts.yml  
drwxr-xr-x.  2 root root  74 Aug 30 17:53 openshift-glusterfs  
drwxr-xr-x.  2 root root  57 Aug 30 17:53 openshift-master  
drwxr-xr-x.  2 root root  84 Aug 30 17:53 openshift-node  
drwxr-xr-x.  2 root root  23 Aug 30 17:53 openshift-preflight  
-rw-r--r--.  1 root root 583 Aug  4 07:46 README.md  
-rw-r--r--.  1 root root 584 Aug  4 07:46 rhel_subscribe.yml  
lrwxrwxrwx.  1 root root   1 Aug 30 17:53 roles -> ../../roles  
-rw-r--r--.  1 root root  57 Aug  4 07:46 vagrant.yml
```

2.4. Inspect the **config.yml** installation playbook file.

Use the **cat** command to display the file.

```
[root@console ~]# cat \  
/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml  
---  
- include: openshift-cluster/config.yml
```

The file does not run any modules, but includes another playbook located in the **openshift-cluster** directory.

3. Review the network configuration.

- 3.1. All the VMs in the environment have the same base network configuration, that is, a main **eth0** network device with an IP address in the **172.25.250.0/24** network range.

Run the **ip** command to review the network configuration on the **console** VM.

```
[root@console ~]# ip a  
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1  
      link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```

        inet 127.0.0.1/8 scope host lo
            valid_lft forever preferred_lft forever
        inet6 ::1/128 scope host
            valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
        link/ether 52:54:00:00:fa:0a brd ff:ff:ff:ff:ff:ff
        inet 172.25.250.10/24 brd 172.25.250.255 scope global eth0
            valid_lft forever preferred_lft forever
        inet6 fe80::5054:ff:fe00:fa0a/64 scope link
            valid_lft forever preferred_lft forever

```

3.2. Exit the **console** VM and log in to the **master1** VM as the **root** user.

```

[root@console ~]# exit
[student@workstation ~]$ ssh root@master1
[root@master1 ~]#

```

3.3. Run the **ip a** command to review the network devices.

Notice the network device **docker0**, which is the Docker service bridge that containers use.

```

[root@master1 ~]# ip a
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN qlen 1
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 scope host lo
        valid_lft forever preferred_lft forever
    inet6 ::1/128 scope host
        valid_lft forever preferred_lft forever
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    qlen 1000
        link/ether 52:54:00:00:fa:0b brd ff:ff:ff:ff:ff:ff
        inet 172.25.250.11/24 brd 172.25.250.255 scope global eth0
            valid_lft forever preferred_lft forever
        inet6 fe80::5054:ff:fe00:fa0b/64 scope link
            valid_lft forever preferred_lft forever
3: docker0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state DOWN
    link/ether 02:42:fb:fb:7a:4b brd ff:ff:ff:ff:ff:ff
    inet 172.17.0.1/16 scope global docker0
        valid_lft forever preferred_lft forever

```

3.4. Log in to **console** VM.

Use the **ssh** command to log in to the **console** VM. When prompted, use **redhat** as the password.

```

[root@master1 ~]# ssh console
root@console's password: redhat

```

4. Review the DNS requirements.

4.1. Run the **dig** command against **master1.lab.example.com**. The output allows you to ensure that DNS names are resolved by the DNS server.

```
[root@console ~]# dig master1.lab.example.com | grep "IN"
;master1.lab.example.com. IN A
master1.lab.example.com. 3600 IN A 172.25.250.11
```

- 4.2. To repeat the previous command for all nodes in the environment, create a Bash array.



Note

You can also run the script located at [/root/D0380/labs/cluster-provision/test-dig.sh](#).

```
[root@console ~]# declare -a nodes=(console services \
master1 master2 master3 \
infra1 infra2 \
apps1 apps2)
```

- 4.3. Run the **dig** command for each entry in the loop. Ensure that the output contains an entry comprised of the node's fully qualified domain name and an IP address.

```
[root@console ~]# for node in ${nodes[@]}; \
do dig ${node}.lab.example.com | grep "IN"; done
;console.lab.example.com. IN A
console.lab.example.com. 3600 IN A 172.25.250.10
;services.lab.example.com. IN A
services.lab.example.com. 3600 IN A 172.25.250.40
;master1.lab.example.com. IN A
master1.lab.example.com. 3600 IN A 172.25.250.11
;master2.lab.example.com. IN A
master2.lab.example.com. 3600 IN A 172.25.250.12
;master3.lab.example.com. IN A
master3.lab.example.com. 3600 IN A 172.25.250.13
;infra1.lab.example.com. IN A
infra1.lab.example.com. 3600 IN A 172.25.250.21
;infra2.lab.example.com. IN A
infra2.lab.example.com. 3600 IN A 172.25.250.22
;apps1.lab.example.com. IN A
apps1.lab.example.com. 3600 IN A 172.25.250.31
;apps2.lab.example.com. IN A
apps2.lab.example.com. 3600 IN A 172.25.250.32
```

5. Ensure connectivity to the nodes.

Run the Ansible **ping** ad hoc command from the **console** VM against the **[OSEv3]** group. The hosts inventory file is located at [/root/D0380/labs/cluster-provision/hosts-ucf](#).

Success for this command indicates two things: that network connectivity is fully functional, and that SSH passwordless access is properly configured. Because Ansible requires SSH keys to be installed on the hosts that it manages, ensure that all hosts return a **SUCCESS**.

```
[root@console ~]# cd /root/D0380/labs/cluster-provision/
[root@console cluster-provision]# ansible -i hosts-ucf OSEv3 -m ping
master2.lab.example.com | SUCCESS => {
```

```
        "changed": false,
        "ping": "pong"
    }
master3.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
infra1.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
master1.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
infra2.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
infra3.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
apps1.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
apps2.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
console.lab.example.com | SUCCESS => {
    "changed": false,
    "ping": "pong"
}
```

This concludes the guided exercise.

Executing Pre-installation Tasks

Objective

After completing this section, students should be able to discuss the required tasks before running the advanced installation method.

Preparing Hosts for OpenShift Container Platform

Before installing OpenShift Container Platform, either using the advanced or quick installation method, a number of prerequisite steps are required to ensure that all systems are compatible and ready to run the installation.

The following is a list of steps to perform on the hosts to ensure that the cluster is ready for the installation of OpenShift Container Platform with the advanced Ansible installation method.

- Propagate SSH keys from the masters to all the nodes for passwordless access.
- Enable the required repositories on the systems.
- Ensure that the subscription manager is running.
- Install and configure the Docker service.
- Configure the storage for Etcd.
- Ensure that the **NetworkManager** service is enabled. Run the **systemctl status NetworkManager** command on all hosts to retrieve the status of the service.
- Set SELinux to **enforcing** mode.
- Install the base packages required by the installer:
 - *wget*
 - *git*
 - *net-tools*
 - *bind-utils*
 - *iptables-services*
 - *bridge-utils*
 - *bash-completion*
 - *kexec-tools*
 - *sos*
 - *psacct*
 - *atomic-openshift-docker-excluder*

- *atomic.openshift-excluder*

SSH Keys Propagation

The Ansible installer needs to use the **ssh** command to connect to all of the hosts without using a password. Use the **ssh-copy-id** utility to propagate SSH keys. Repeat the command for each host in the cluster.

```
[root@demo ~]# ssh-copy-id -i ~/.ssh/PUBLIC KEY.pub REMOTE SERVER
```

Repository Configuration

Enable the required repositories on the systems to prepare them for the installation of OpenShift Container Platform. Each host on the OpenShift cluster (masters and nodes) must be registered using Red Hat Subscription Management (*RHSM*) and attached to the valid OpenShift Container Platform subscriptions. To register and attach the hosts, use the **subscription-manager register** and **subscription-manager attach** commands.

The **rhel-7-server-ose-3.6-rpms** repository provides all OpenShift Container Platform packages. To enable the required repositories, use the **subscription-manager repos --enable** command.

Run the **yum repolist enabled** command to ensure that the following repositories are enabled:

- **rhel-7-server-rpms**
- **rhel-7-server-extras-rpms**
- **rhel-7-server-ose-3.6-rpms**
- **rhel-7-fast-datapath-rpms**

Run the following steps to enable only the required repositories:

1. Disable all repositories.

```
[root@demo ~]# yum-config-manager --disable "*"
```

2. Enable the required repositories.

```
[root@demo ~]# yum-config-manager \
--enable "rhel_dvd" \
--enable "rhel-7-server-extras-rpms" \
--enable "rhel-7-server-updates-rpms" \
--enable "rhel-7-server-ose-3.6-rpms" \
--enable "rhel-7-fast-datapath-rpms"
```

Before running the installer, administrators must ensure that all packages are up-to-date on the system. The subscription manager must also be running. To retrieve the status of the subscription manager, run the **subscription-manager status** command.

```
[root@demo ~]# subscription-manager status
registered to: subscription.rhn.redhat.com
```

```
server type: subscription management service
subscription-manager: 1.1.2-1.el6
python-rhsm: 1.1.3-1.el6
```



Note

This classroom environment has access to offline repositories; consequently, the systems are not registered.

Docker Configuration

Before running the installer, administrators must configure Docker on all master and node hosts. OpenShift Container Platform uses Docker to store containers and images. Doing so provides the benefit of configuring the Docker storage options before installing OpenShift Container Platform. This storage is not persistent and is separated from the persistent storage allocated to applications. For Red Hat Enterprise Linux 7 systems, install Docker version **1.12**. On Red Hat Enterprise Linux Atomic Host systems, Docker is already installed, configured, and running by default.

In RHEL Atomic hosts, the Docker service uses a logical volume that is supported for production environments. However, you must ensure that enough space is allocated for this volume, according to the storage requirements.

For RHEL hosts, the default storage is a pool on a set of loopback devices, which is not supported for production environments. You can use the **docker-storage-setup** script (included with the Docker service) to create a storage pool device and configure the storage driver for Red Hat Container Registry.

You can either add an additional block device or use an existing LVM volume group. To use an additional block device, populate the **/etc/sysconfig/docker-storage-setup** file as follows:

```
DEVS=/dev/vdc①
VG=docker-vg②
```

- ^① The device to use on the system.
- ^② The LVM volume group to assign to the Docker service.

To use existing storage, populate the **/etc/sysconfig/docker-storage-setup** file with:

```
VG=docker-vg
```

Run the **docker-storage-setup** command to configure the storage. The utility parses the **/etc/sysconfig/docker-storage-setup** file to configure the storage.

```
[root@demo ~]# docker-storage-setup
0
Checking that no-one is using this disk right now ...
OK

Disk /dev/vdc: 31207 cylinders, 16 heads, 63 sectors/track
sfdisk: /dev/vdc: unrecognized partition table type
```

```
Old situation:  
sfdisk: No partitions found  
  
New situation:  
Units: sectors of 512 bytes, counting from 0  
...
```



Note

Provide an empty partition to create the volume group where all the container images are saved. Optionally, you can use the remaining free space from the volume group where the root file system is located. However, this is not the recommended practice. Also,



Important

Before using Red Hat Container Registry or OpenShift Container Platform, ensure that the **docker-pool** logical volume is large enough to accommodate the infrastructure needs. The volume used for the Docker service should have at least 60% of its capacity free.

After you have configured the storage, ensure that the **docker** service is running. To retrieve the status of the **docker** service use the **systemctl** command.

```
[root@demo ~]# systemctl is-active docker
```

Reinitializing the Docker Service

If Red Hat Container Registry is already running, administrators must reinitialize the service.

To reinitialize the Docker service:

1. Stop the daemon.

```
[root@demo ~]# systemctl stop docker
```

2. Delete the files in the **/var/lib/docker** directory, to minimize conflicts with existing container images.

```
[root@demo ~]# rm -rf /var/lib/docker/*
```

3. Restart the daemon.

```
[root@demo ~]# systemctl restart docker
```

Configuring Etcd

Etcd is a distributed key-value store for distributed systems. OpenShift uses it to store system configuration and state. OpenShift Container Platform installs and configures Etcd, but you must configure storage for the service prior to its installation.

To configure the storage for Etcd, administrators must ensure that the **/var/lib/etcd** directory is used by the service and it meets the minimum disk space storage requirement (20GB). To improve the performance, administrators can use a separate disk for Etcd back end. For example, you can create an LVM logical volume and mount it on the **/var/lib/etcd** share.

Using Ansible for Pre-installation Tasks

You can perform the prerequisite steps manually or using a set of scripts, but Red Hat encourages using Ansible. This provides the benefit of using an automation engine that centralizes tasks, which can be managed via a *Version Control System (VCS)*, such as Subversion or Git.

The following playbook contains the pre-installation tasks.

```
tasks:
  - name: Deploy SSH key to root at all nodes①
    authorized_key:
      user: root
      key: "{{ lookup('file', '/root/.ssh/lab_rsa.pub') }}"

  - name: Install docker②
    yum:
      name: docker
      state: latest

  - name: Customize /etc/sysconfig/docker-storage-setup③
    copy:
      src: files/docker-storage-setup
      dest: /etc/sysconfig/docker-storage-setup

  - name: Verify presence of /dev/docker-vg/docker-pool④
    stat:
      path: /dev/docker-vg/docker-pool
    register: docker_vg_status

  - name: Run docker-storage-setup⑤
    command: /usr/bin/docker-storage-setup
    when: docker_vg_status.stat.islnk is not defined

  - name: Start and enable docker⑥
    service:
      name: docker
      state: started
      enabled: true

  - name: Install required packages⑦
    yum:
      name: "{{ item }}"
      state: latest
    with_items:
      - wget
      - git
      - net-tools
      - bind-utils
      - iptables-services
      - bridge-utils
      - bash-completion
      - kexec-tools
```

```

- sos
- psacct
- atomic-openshift-docker-excluder
- atomic-openshift-excluder

- name: Install OpenShift tools⑧
yum:
  name: atomic-openshift-utils
  state: latest
  when: inventory_hostname in groups['masters']

- name: Configure Etcd⑨
  include_role:
    name: make-lvol
  vars:
    name: '{{ vol_name }}'
    pv_list: '{{ device }}'
    mount_point: '{{ mt}}'
  when: inventory_hostname in groups['etcd']

```

- ① Deploys the SSH keys to all hosts in the **master**'s group from an inventory
- ② Installs Red Hat Container Registry using the Yum installer.
- ③ Customizes the volume group configuration for the Docker service.
- ④ Verifies that the **docker-pool** volume group was created
- ⑤ Runs the **docker-storage-setup** script
- ⑥ Starts the **docker** service
- ⑦ Installs packages needed by OpenShift
- ⑧ Installs the *atomic-openshift-utils* package on the **master** host
- ⑨ Creates a logical volume for the Etcd service and mounts it on the **/var/lib/etcd** directory



References

Knowledge base article with a problem in the disconnected installation

| <https://access.redhat.com/solutions/3173341>

Further information is available in the Host Preparation section of the *OpenShift Container Platform 3.6; OpenShift Container Platform Installation and Configuration Guide* at

| <https://access.redhat.com/documentation/en-US/index.html>

Guided Exercise: Executing Pre-installation Tasks

In this guided exercise, you will run a set of pre-installation tasks to prepare the environment for the deployment of OpenShift Container Platform.

Outcomes

You should be able to:

- Inspect the Ansible Playbook used to execute the pre-installation tasks by reviewing the modules it uses.
- Run the pre-installation Ansible Playbook.
- Review the changes applied by the Ansible Playbook on the hosts it was run against.

Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab cluster-pre setup** command. The script creates the files for this exercise in the **/root/** directory of the **console** server. It also installs the *tree* package.

```
[student@workstation ~]$ lab cluster-pre setup
```

Steps

1. Log in to **console** as the **root** user.

On the **workstation** machine, open a terminal and use the **ssh** command to connect to **console**.

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

2. The Ansible Playbook for this exercise is located in the **/root/D0380/labs/cluster-pre** directory. Go to that directory.

```
[root@console ~]# cd /root/D0380/labs/cluster-pre
[root@console cluster-pre]#
```

3. Inspect the files in the directory.

```
[root@console cluster-pre]# tree
.
├── files
│   ├── rhel-7-fast-datapath-rpms.repo
│   ├── rhel-7-server-extras-rpms.repo
│   ├── rhel-7-server-optional-rpms.repo
│   ├── rhel-7-server-ose-3.6-rpms.repo
│   └── rhel-7-server-updates-rpms.repo
└── hosts-ucf
└── make-lvol
    └── tasks
```

```

└── main.yml
└── pre-install.yml

3 directories, 8 files

```

- The **pre-install.yml** file is the Ansible Playbook. It contains the set of required tasks to perform prior to the installation of OpenShift Container Platform.
- The **files** directory contains the ***.repo** Yum repository files, which has been copied to the **/etc/yum.repos.d** directory on each host configured by Ansible.
- The **hosts-ucf** is the inventory file that defines the host in the environment.
- The **make-lvol** directory is an Ansible role used by the playbook that creates LVM volumes.

4. Use a text editor to inspect the **pre-install.yml** Ansible Playbook.

```

---
- hosts: nodes
  become: yes
  vars:
    ...

```

- The first block defines a set of variables used by the tasks in the playbook. The **ip_address** and **hostname** variables are used by the tasks that perform DNS resolution and DNS reverse resolution. The **vol_name**, **device**, and **mt** variables are used by the **make-lvol** module.

```

vars:
  ip_address: 172.25.254.254
  hostname: classroom.example.com
  vol_name: etcd
  device: /dev/vdc
  mt: /var/lib/etcd

```

- The first task invokes the **command** module to run the **host** command. The task uses the **hostname** variable and saves the output in the **dns_lookup** variable.

```

tasks:
  - name: Perform DNS lookup
    command:
      host {{ hostname }}
    register: dns_lookup

```

- The second task invokes the **command** module to run the **host** command. The task uses the **ip_address** variable and saves the output in the **dns_reverse_lookup** variable.

```

  - name: Performing DNS reverse lookup
    command:
      host {{ ip_address }}
    register: dns_reverse_lookup

```

- The next two tasks invoke the **debug** module to display the output of the two variables created by the previous modules, **dns_lookup** and **dns_reverse_lookup**.

- name: DNS lookup result
 debug:
 var: dns_lookup.stdout
- name: DNS reverse lookup result
 debug:
 var: dns_reverse_lookup.stdout

- The next task ensures that SELinux is in **Enforcing** mode.

```
- name: Enable SELinux in Enforcing Mode
  selinux:
    policy: targeted
    state: enforcing
```

- The next task ensures that the **NetworkManager** service is enabled and running.

```
- name: Ensuring NetworkManager is enabled and running
  service:
    name: NetworkManager
    state: started
    enabled: yes
```

- The following five tasks ensure that the OpenShift Container Platform Yum repository files are present on the systems.

```
- name: Checking Fast datapath Yum repositories
  copy:
    dest: /etc/yum.repos.d
    src: rhel7-fast-datapath-rpms.repo
    owner: root
    group: root
    mode: 0644
```

- The next task ensures that all packages are up-to-date.

```
- name: Upgrade all packages
  yum:
    name: '*'
    state: latest
```

- The last task invokes the **make-lvol** module to create a logical volume on the master nodes. To select the nodes to run the module on, it uses the **when** conditional.

```
- name: Configure etcd
  include_role:
    name: make-lvol
  vars:
    name: '{{ vol_name }}'
    pv_list: '{{ device }}'
```

```
mount_point: '{{ mt }}'
when: inventory_hostname in groups['etcd']
```

- 4.1. Exit the text editor.
5. Run the **ansible-playbook** command against the playbook. Use the **-i hosts-ucf** option to use the classroom inventory file.

Review carefully the tasks as they are being executed. Ensure that the **PLAY RECAP** section does not display any **failed** tasks.

```
[root@console cluster-pre]# ansible-playbook -i hosts-ucf pre-install.yml
PLAY [nodes] *****
...output omitted...

TASK [Gathering Facts] *****
ok: [master3.lab.example.com]
ok: [infra2.lab.example.com]
ok: [master2.lab.example.com]
ok: [master1.lab.example.com]
ok: [infra1.lab.example.com]
ok: [apps1.lab.example.com]
ok: [infra3.lab.example.com]
ok: [apps2.lab.example.com]
...
PLAY RECAP *****
apps1.lab.example.com : ok=13    changed=3      unreachable=0      failed=0
apps2.lab.example.com : ok=13    changed=3      unreachable=0      failed=0
infra1.lab.example.com : ok=13    changed=3      unreachable=0      failed=0
infra2.lab.example.com : ok=13    changed=3      unreachable=0      failed=0
infra3.lab.example.com : ok=13    changed=3      unreachable=0      failed=0
master1.lab.example.com: ok=19    changed=8      unreachable=0      failed=0
master2.lab.example.com: ok=19    changed=8      unreachable=0      failed=0
master3.lab.example.com: ok=19    changed=8      unreachable=0      failed=0
```

6. Review the operating system service changes performed by the playbook.
- 6.1. From the **console** VM, run the **getenforce** command to retrieve the SELinux state.

```
[root@console cluster-pre]# getenforce
Enforcing
```

- 6.2. Ensure that the **NetworkManager** service is running and enabled on the **console** VM.

```
[root@console cluster-pre]# systemctl status NetworkManager.service
● NetworkManager.service - Network Manager
   Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled;
             vendor preset: enabled)
     Active: active (running) since Tue 2017-09-05 13:54:19 PDT; 5 days ago
       Docs: man:NetworkManager(8)
   Main PID: 506 (NetworkManager)
     CGroup: /system.slice/NetworkManager.service
             └─506 /usr/sbin/NetworkManager --no-daemon
...
...
```

- 6.3. Ensure that the **NetworkManager** service is running and enabled on the **master1** VM.

```
[root@console cluster-pre]# ssh master1 systemctl status NetworkManager.service
● NetworkManager.service - Network Manager
  Loaded: loaded (/usr/lib/systemd/system/NetworkManager.service; enabled;
  vendor preset: enabled)
    Active: active (running) since Tue 2017-09-05 13:55:57 PDT; 5 days ago
      Docs: man:NetworkManager(8)
   Main PID: 567 (NetworkManager)
     Memory: 7.2M
        CGroup: /system.slice/NetworkManager.service
                  └─567 /usr/sbin/NetworkManager --no-daemon
```

7. Review the LVM updates performed by the Ansible Playbook.

7.1. On the **workstation** VM, use the **ssh** command to log in to the **master1** VM.

```
[root@console cluster-pre]# ssh master1
[root@master1 ~]#
```

7.2. Run the **lvdisplay** command to ensure that an **etcd-lv** logical volume exists.

```
[root@master1 ~]# lvdisplay
...
--- Logical volume ---
LV Path          /dev/etcd-vg/etcd-lv
LV Name          etcd-lv
VG Name          etcd-vg
LV UUID          SMvwHO-IBzj-pKaj-WAGc-3qNZ-8BHQ-MWiQd0
LV Write Access  read/write
LV Creation host, time master1.lab.example.com, 2017-09-05 11:53:18 -0700
LV Status        available
# open           1
LV Size          1020.00 MiB
Current LE       255
Segments         1
Allocation       inherit
Read ahead sectors auto
- currently set to 8192
Block device    252:0
--- Logical volume ---
LV Name          docker-pool
VG Name          docker-vg
LV UUID          wNtqXF-tT5D-f5ZB-r91p-C60o-qNou-bNFcdV
LV Write Access  read/write
LV Creation host, time master1.lab.example.com, 2017-11-09 18:53:35 -0500
LV Pool metadata docker-pool_tmeta
LV Pool data     docker-pool_tdata
LV Status        available
# open           0
LV Size          <7.95 GiB
Allocated pool data 0.24%
Allocated metadata 0.16%
Current LE       2035
Segments         1
Allocation       inherit
Read ahead sectors auto
- currently set to 8192
Block device    252:2
```

- 7.3. Ensure that the volume is mounted on the **/var/lib/etcd** directory.

```
[root@master1 ~]# mount | grep etcd
/dev/mapper/etcd--vg-etcd--lv on /var/lib/etcd type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
```

- 7.4. Use the **ssh** command to connect to the **master2** VM as the **root** user. When prompted, use **redhat** as the password.

```
[root@master1 ~]# ssh root@master2
root@master2's password: redhat
```

- 7.5. Run the **mount** command to ensure that the **etcd-lv** volume is mounted on **/var/lib/etcd**.

```
[root@master2 ~]# mount | grep etcd
/dev/mapper/etcd--vg-etcd--lv on /var/lib/etcd type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
```

- 7.6. Use the **ssh** command to connect to **master3** as the **root** user. When prompted, use **redhat** as the password.

```
[root@master2 ~]# ssh root@master3
root@master3's password: redhat
```

- 7.7. Run the **mount** command to ensure that the **etcd-lv** is mounted on **/var/lib/etcd**.

```
[root@master3 ~]# mount | grep etcd
/dev/mapper/etcd--vg-etcd--lv on /var/lib/etcd type xfs
(rw,relatime,seclabel,attr2,inode64,noquota)
```

This concludes the guided exercise.

Running the Advanced Installation Method

Objectives

After completing this section, students should be able to:

- Describe the Ansible advanced installation method.
- Review the steps that the advanced installer performs.
- Troubleshoot an advanced installation.
- Identify the post-installation tasks required for a validation of OpenShift Container Platform.
- Run the advanced installation method to build the HA cluster.

Introduction to the Advanced Installation Method

Customize OpenShift Container Platform with the Ansible advanced installation method. Unlike the quick installation method, the advanced installation method is not interactive. It uses an inventory file with configuration options to determine how to install the OpenShift cluster. Both methods are supported for production deployments, however, the advanced installation gives administrators more control over their cluster's configuration.



Note

Administrators who wish to use the advanced installation method must be familiar with Ansible.

Ensure that the environment meets the prerequisites covered in *the section called “Provisioning Servers”* before installing OpenShift Container Platform. The section includes the systems requirements and the required Docker service configuration.

Configuring the Ansible Inventory File

The advanced installation method uses an Ansible inventory file to detect the environment configuration. For example, an **nfs** group in the inventory files describes the servers that run the Red Hat Container Registry upon installation.

The **[OSEv3:vars]** group contains the required variables, such as the network topology, or the label selectors to use. The **nodes** hosts group defines the nodes to install, and contains variables such as label selectors.

This inventory file describes the base installation to run. Replace the content of the file with the desired configuration before invoking the Ansible installer with this inventory file. The default inventory file contains comments that describe the various options.



Note

Consult the references at the end of the section for a link to a default Ansible inventory file.

Many of the available variables are optional, and do not need to be set. The default values should be suitable for development environments, but you should be familiar with all options before deploying OpenShift Container Platform in a production environment. Doing so gives you a better understanding on how to manage the cluster's life cycle.

Although not exhaustive, the following table describes the variables that can be set in the **[OSEv3:vars]** section.

Ansible Inventory File Variables	
Variable	Purpose
ansible_ssh_user	Sets the SSH user for the installer to use. Defaults to root . If using SSH key-based authentication, the key should be managed by an SSH agent.
containerized	If set to true , containerized OpenShift Container Platform services, such as the integrated registry, run on all target master and node hosts in the cluster instead of installed using RPM packages. If set to false or unset, the default RPM method is used.
openshift_rolling_restart_mode	Enables rolling restarts of the masters (master nodes are taken down one at a time) when running the upgrade playbook directly. Defaults to services , which allows rolling restarts of services on the masters. If set to system , it enables rolling, full-system restarts and also works for single master clusters.
os_sdn_network_plugin_name	Configures the OpenShift SDN plug-in to use for the pod network, which defaults to redhat/openshift-ovs-subnet for the standard SDN plug-in.
openshift_master_named_certificates	Defines custom security certificates that are used during the installation phase.
openshift_portal_net	Defines the subnet in which services are created within the OpenShift Container Platform SDN. The network block should be private and must not conflict with any existing network blocks in the infrastructure to which pods, nodes, or the master may require access to. Such a conflict would cause the installation to fail.

Ansible Inventory File Variables

openshift_docker_additional_registries	Specifies the additional registries that OpenShift Container Platform should add to the Docker service configuration.
openshift_hosted_metrics_public_url	Sets the host name for integration with the metrics console by overriding the metricsPublicURL variable in the master configuration for cluster metrics. Ensure that the host name is reachable via the router.

The inventory file contains many groups. Each group describes a role in the cluster. The following list describes the groups present in an inventory file.

- **[masters]**: Defines the servers that run master services. For example:

```
master1.lab.example.com openshift_ip=172.25.250.11 openshift_public_ip=172.25.250.11
```

- **[etcd]**: Defines the servers that run the Etcd services. For example:

```
master[1:3].lab.example.com
```

- **[nodes]**: Defines the servers that run docker containers. You can use variables to define labels. For example:

```
master1.lab.example.com openshift_node_labels="{'region': 'master'}" openshift_schedulable=false openshift_ip=172.25.250.11  
openshift_public_ip=172.25.250.11
```

- **[nfs]**: Defines the servers that run the NFS service for the physical volumes. For example:

```
console.lab.example.com
```

- **[lb]**: Defines the group for HAProxy as the load balancer. For example:

```
console.lab.example.com containerized=false
```



Note

A recommended practice consists in using a *Version Control System (VCS)*, such as Subversion or Git, to keep track of the changes in the inventory file. You can revert the inventory file to a previous stable version if changes inadvertently break the installation.



Note

The classroom environment provides an example inventory file in the **openshift-install** Git repository on the **services** machine. This inventory file contains the variables required by the advanced installation method. Use it as a reference for all the possible advanced installation variables and host declarations.

Running the Advanced Installation Using Ansible

Run the advanced installation after populating the Ansible inventory file according to your environment. OpenShift Container Platform uses a set of task, files, modules, and variables to deploy the cluster. Ansible uses idempotent tasks, which means that the same task can be run multiple times on the systems without disruption. It also provides a human-readable format of the infrastructure, which speeds up configuration and provides ease of maintenance. The advanced method reduces the complexity of configuration, because preset values are used to get a cluster up and running, which can also be customized later on.

Ansible provides the output of each task it runs in real time, such as creating users, defining user roles, or configuring the Red Hat Container Registry. When the installation completes, a summary is displayed, that indicates how many tasks succeeded and how many tasks failed.

```
...
PLAY [Create initial host groups for localhost] ****
TASK [include_vars] ****
ok: [localhost]
PLAY [Populate config host groups] ****
TASK [Evaluate groups - g_etcd_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_master_hosts or g_new_master_hosts required] ***
skipping: [localhost]
TASK [Evaluate groups - g_node_hosts or g_new_node_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_lb_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_nfs_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_nfs_hosts is single host] ****
skipping: [localhost]
TASK [Evaluate groups - g_glusterfs_hosts required] ****
skipping: [localhost]

...
PLAY RECAP ****
apps1.lab.example.com      : ok=237    changed=11    unreachable=0      failed=0
apps2.lab.example.com      : ok=237    changed=11    unreachable=0      failed=0
console.lab.example.com   : ok=125    changed=3     unreachable=0      failed=0
infra1.lab.example.com    : ok=237    changed=11    unreachable=0      failed=0
infra2.lab.example.com    : ok=237    changed=11    unreachable=0      failed=0
infra3.lab.example.com    : ok=237    changed=11    unreachable=0      failed=0
localhost                  : ok=14     changed=0     unreachable=0      failed=0
master1.lab.example.com   : ok=685    changed=48    unreachable=0      failed=0
master2.lab.example.com   : ok=445    changed=30    unreachable=0      failed=0
```

You can either update the inventory file or diagnose the environment and re-execute the playbook if a task fails. There are common errors that prevent the playbook from completing, such as:

- Insufficient memory.
- Insufficient disk space.
- Misconfigured or missing repositories.
- Lack of external connectivity for the Docker service.
- Unreachable nodes.

The **openshift_disable_check** variable can be used to skip certain tests from being performed by the installer, thus limiting potential installation failures. For example, to disable the Docker service and image checks and resources checks such as storage and memory availability, define the variable in the **[OSEv3:vars]** group as follows:

```
openshift_disable_check=disk_availability,docker_storage,memory_availability,
docker_image_availability
```

OpenShift Container Platform Deployment Phases

The Ansible Playbook does not contain all the deployment tasks; in fact, this is a template file that includes another playbook, located in the **openshift-cluster** directory.

The modular structure breaks the installation phases down into a number of modules and tasks, available in various directories, such as **common** and **adhoc**.

```
---
- include: openshift-cluster/config.yml
```

The **config.yml** playbook file in the **openshift-cluster** directory initializes a set of deployments, as follows:

```
---
- include: initialize_groups.yml
  tags:
  - always

- include: ../../common/openshift-cluster/std_include.yml
  tags:
  - always

- include: ../../common/openshift-cluster/config.yml
  vars:
    openshift_cluster_id: "{{ cluster_id | default('default') }}"
    openshift_debug_level: "{{ debug_level | default(2) }}"
    openshift_deployment_subtype: "{{ deployment_subtype | default(None) }}"
```

Ansible parses all the playbooks and runs various tasks for the installation, which includes:

- Ensuring system packages, such as Python, match the requirements:

```
TASK [openshift_facts : Validate python version]*****
skipping: [master3.demo.example.com]
```

```

skipping: [master2.demo.example.com]
skipping: [master1.demo.example.com]
skipping: [apps2.demo.example.com]
skipping: [infra3.demo.example.com]
skipping: [apps1.demo.example.com]
skipping: [infra2.demo.example.com]
skipping: [infra1.demo.example.com]
skipping: [console.demo.example.com]
...

```

- Ensuring that package dependencies are installed:

```

TASK [openshift_facts : Ensure various deps are installed] ****
ok: [master3.demo.example.com] => (item=yum-utils)
ok: [infra1.demo.example.com] => (item=yum-utils)
ok: [infra2.demo.example.com] => (item=PyYAML)
ok: [infra1.demo.example.com] => (item=PyYAML)
ok: [master2.demo.example.com] => (item=PyYAML)
ok: [master1.demo.example.com] => (item=PyYAML)
ok: [master3.demo.example.com] => (item=PyYAML)
ok: [infra1.demo.example.com] => (item=python-dbus)
ok: [master3.demo.example.com] => (item=python-dbus)
ok: [master2.demo.example.com] => (item=python-dbus)
ok: [master1.demo.example.com] => (item=python-dbus)
...

```

- Setting Red Hat Container Registry facts:

```

TASK [openshift_docker_facts : Set docker facts] ****
ok: [master1.demo.example.com] => (item={'local_facts': {'blocked_registries': u'registry.access.redhat.com, docker.io', 'hosted_registry_insecure': False, 'log_driver': u'', 'disable_push_dockerhub': u'', 'selinux_enabled': u'', 'additional_registries': u'registry.demo.example.com:5000', 'hosted_registry_network': u'172.30.0.0/16', 'log_options': u'', 'insecure_registries': u'registry.demo.example.com:5000', 'options': u'', 'use_system_container': False}, 'role': u'docker'})
ok: [master1.demo.example.com] => (item={'local_facts': {'sdn_mtu': u''}, 'role': u'node'})
...

```

- Configuring the firewall:

```

TASK [os_firewall : Start and enable firewalld service] ****
skipping: [master1.lab.example.com]
...

```

- Configuring the registry:

```

TASK [docker : Set registry params] ****
skipping: [master1.demo.example.com] => (item={'reg_conf_var': u'ADD_REGISTRY', 'reg_flag': u'--add-registry', 'reg_fact_val': [u'registry.access.redhat.com:5000', 'u'registry.access.redhat.com']}))
skipping: [master1.demo.example.com] => (item={'reg_conf_var': u'BLOCK_REGISTRY', 'reg_flag': u'--block-registry', 'reg_fact_val': [u'registry.access.redhat.com', 'u'docker.io']}))
skipping: [master1.demo.example.com] => (item={'reg_conf_var': u'INSECURE_REGISTRY', 'reg_flag': u'--insecure-registry', 'reg_fact_val': [u'registry.access.redhat.com:5000']}))

```

```
...
```

- Configuring certificates:

```
TASK [etcd_server_certificates : Sign and create the server crt] *****
changed: [master2.demo.example.com -> master1.demo.example.com]
changed: [master3.demo.example.com -> master1.demo.example.com]
changed: [master1.demo.example.com -> master1.demo.example.com]

TASK [etcd_server_certificates : Create the peer csr] *****
changed: [master1.demo.example.com -> master1.demo.example.com]
changed: [master2.demo.example.com -> master1.demo.example.com]
changed: [master3.demo.example.com -> master1.demo.example.com]
...

```

- Configuring NFS:

```
TASK [openshift_storage_nfs : remove exports from /etc(exports)] *****
ok: [console.demo.example.com] => (item=/exports/registry *(rw,root_squash))
ok: [console.demo.example.com] => (item=/exports/metrics *(rw,root_squash))
ok: [console.demo.example.com] => (item=/exports/logging-es *(rw,root_squash))
ok: [console.demo.example.com] => (item=/exports/logging-es-ops *(rw,root_squash))
ok: [console.demo.example.com] => (item=/exports/etc *(rw,root_squash))
...

```

- Configuring cluster permissions:

```
TASK [openshift_manageiq : Configure role/user permissions] *****
ok: [master1.demo.example.com] => (item={u'resource_kind': u'role', u'resource_name': u'admin', u'user': u'management-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'role', u'resource_name': u'management-infra-admin', u'user': u'management-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'cluster-role', u'resource_name': u'cluster-reader', u'user': u'system:serviceaccount:management-infra:management-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'scc', u'resource_name': u'privileged', u'user': u'system:serviceaccount:management-infra:management-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'cluster-role', u'resource_name': u'system:image-puller', u'user': u'system:serviceaccount:management-infra:inspector-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'scc', u'resource_name': u'privileged', u'user': u'system:serviceaccount:management-infra:inspector-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'cluster-role', u'resource_name': u'self-provisioner', u'user': u'system:serviceaccount:management-infra:management-admin'})
ok: [master1.demo.example.com] => (item={u'resource_kind': u'cluster-role', u'resource_name': u'hawkular-metrics-admin', u'user': u'system:serviceaccount:management-infra:management-admin'})
...

```

Reviewing the OpenShift Container Platform Installation

After the installation completes without errors, the cluster should be ready to run applications. There are some basic checks that you can perform to ensure that the environment can safely run applications and that the cluster is stable, such as:

- Ensuring that DNS resolution is fully functional for all nodes in the cluster

- Verifying that master nodes are not marked as **schedulable**.
- Ensuring that application nodes are marked as **ready**
- Ensuring that all router endpoints are up
- If monitoring is deployed, ensuring that the metric endpoints are reachable
- If a logging platform such as Kibana is deployed, ensuring that the service is up
- If the Fluentd data collector is configured, checking the status of the Fluentd pods
- Testing cluster roles such as the cluster **binding roles**
- To give access to the cluster to external hosts, set the adequate policy



Important

This list does not cover all possibilities. You need to review the items defined in the inventory file before installing OpenShift Container Platform.

Reviewing Configuration Files

Another way to review the installation is to read the various configuration files that were created by the installer. The majority of the configuration files are YAML files, in which the configuration is organized into sections. This gives you the capacity to quickly review the files and determine which keys determine which settings.

On master nodes, configuration files are stored in **/etc/origin/master**. The directory contains security certificates, OAuth, general configuration files, and so on. On the nodes, configuration files are stored in **/etc/origin/node**.

Performing Post-installation Tasks

After you have installed the cluster, you need to perform some post-installation tasks, according to the needs of the organization. For example, you need to configure the adequate set of policies if specific privileges are required. Install supplementary images if you need to integrate a monitoring solution such as Hawkular.

Although not exhaustive, the following list describes post-installation tasks.

- You must reconfigure the image streams if the cluster is running in disconnected mode and does not have access to the Internet. This procedure involves deleting all existing image streams and importing image streams that contain the updated Docker endpoints:

```
[root@demo ~]# oc delete is -n openshift --all
[root@demo ~]# oc create \
-f /usr/share/openshift/examples/image-streams/image-streams-rhel7.json \
-n openshift
```

- Update the registry console to use the internal Red Hat Container Registry in a disconnected environment.

```
[root@demo ~]# oc set image --source=docker \
dc/registry-console \
```

```
registry-console=registry server:5000/openshift3/registry-console:version
```

- Configure the identity provider if the organization uses an IdM such as LDAP or Active Directory.
- Configure logs aggregation to optimize the disk usage.
- To collect the metrics exposed by Kubelet, administrators can enable cluster metrics.



Note

Set the cluster metrics in the Ansible inventory file prior to the cluster installation.

Troubleshooting a Cluster Installation

You can use various tools to troubleshoot the cluster, from the **oc** command to retrieve the active nodes to the more advanced tool, **oc adm**.

The **oc adm diagnostics** command runs a series of checks, looking for errors in the cluster, such as:

- Ensuring that the default registry and router are running and are correctly configured
- Ensuring that **ClusterRoleBindings** and **ClusterRoles** are consistent with the base policy
- Ensuring that all of the client configuration contexts are valid
- Ensuring that SkyDNS is working properly and the pods have connectivity (SkyDNS is a distributed service for announcement and discovery of services built on top of Etcd)
- Validating masters and nodes configuration on the hosts
- Ensuring that the nodes are running and that they are available
- Checking that the **systemd** units are configured as expected for the host

Ansible-based Health Checks

Run additional health checks, available through the Ansible-based tooling, to manage and configure OpenShift Container Platform. The tool can report common deployment problems and uses either the **ansible-playbook** command, or the containerized version of *openshift-ansible*.

Although not exhaustive, the following health checks are a set of diagnostic tasks designed to be run against the Ansible inventory file for a deployed OpenShift Container Platform cluster, using the provided **health.yml** file.

Ansible-based Health Checks	
Check	Purpose
etcd_imagedata_size	Measures the total size of OpenShift Container Platform image data in an Etcd cluster. The check fails if the calculated size exceeds a user-defined limit.

Ansible-based Health Checks	
	If no limit is specified, the check fails if the size of the image data amounts to 50% or more of the currently used space in the Etcd cluster.
etcd_traffic	Detects higher-than-normal traffic on an Etcd host. Fails if a journalctl log entry with an Etcd sync duration warning is found.
docker_storage	Only runs on hosts that depend on the docker daemon. Ensures that Docker service's total usage does not exceed a user-defined limit. If no user-defined limit is set, the Docker service's maximum usage threshold defaults to 90% of the total size available.
logging_index_time	Detects higher-than-normal time delays between log creation and log aggregation by Elasticsearch in a logging stack deployment. Fails if a new log entry cannot be queried through Elasticsearch within a set timeout. Only runs if logging is enabled.

You can run these tests either via the **ansible-playbook** command, or via the **docker** CLI.

To run the health checks using the **ansible-playbook** command, specify the cluster's inventory file and run the **health.yml** playbook:

```
[root@demo ~]# ansible-playbook -i inventory_file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshiftchecks/health.yml
```

Use the **-e** option to set variables in the command line:

```
[root@demo ~]# ansible-playbook -i inventory_file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshiftchecks/health.yml \
-e openshift_check_logging_index_timeout_seconds=45 \
-e etcd_max_image_data_size_bytes=40000000000
```

Use the **docker** command and specify both the cluster's inventory file and the **health.yml** playbook to run the health checks in a Linux container. You can run the command as a non-privileged user, however, it requires that it has the permission to run privileged containers.

```
[user@demo ~]$ docker run -u $(id -u) \①
-v $HOME/.ssh/id_rsa:/opt/app-root/src/.ssh/id_rsa:Z,ro \②
-v /etc/ansible/hosts:/tmp/inventory:ro \③
-e INVENTORY_FILE=/tmp/inventory \
-e PLAYBOOK_FILE=playbooks/byo/openshift-checks/health.yml \④
-e OPTS="-v -e openshift_check_logging_index_timeout_seconds=45 \
-e etcd_max_image_data_size_bytes=40000000000" \⑤
openshift3/ose-ansible
```

- ① Makes the container run with the same UID as the current user, which is required for permissions so that the SSH key can be read inside the container.

- ② Mounts SSH keys as a volume under `/opt/app-root/src/.ssh` under normal usage when running the container as a non-root user.
- ③ Changes `/etc/ansible/hosts` to the location of the cluster's inventory file, if different. This file is bind-mounted to the `/tmp/inventory` directory, which is used according to the `INVENTORY_FILE` environment variable in the container.
- ④ The `PLAYBOOK_FILE` environment variable is set to the location of the `health.yml` playbook relative to `/usr/share/ansible/openshift-ansible` inside the container.
- ⑤ Sets any variables desired for a single run with the `-e key=value` format.



References

Example of an inventory file

<https://github.com/openshift/openshift-ansible/blob/master/inventory/hosts.example>

Further information is available in the Advanced Installation section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at

<https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Diagnostics Tool section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Cluster Administration Guide* at

<https://access.redhat.com/documentation/en-US/index.html>

Guided Exercise: Using the Advanced Installation Method

In this guided exercise, you will use the OpenShift Container Platform advanced installation method and ensure that you can log in to the cluster.

Outcomes

You should be able to:

- Run the advanced installation method playbook.
- Review the installation of OpenShift Container Platform.
- Review OpenShift security certificates.

Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab advanced-installation setup** command. The script installs the required files for this exercise and ensures that the security certificates are present.

```
[student@workstation ~]$ lab advanced-installation setup
```

Steps

1. Inspect the Ansible inventory file created in the previous section.



Note

The lab script created a default **hosts-ucf** inventory on **console** in the **/root/D0380/solutions/cluster-install** directory. If you did not create an inventory file in the previous section, use the file in this directory.

From the **workstation** VM, log in to the **console** VM as the **root** user.

```
[student@workstation ~]$ ssh root@console
```

- 1.2. Pull the Git repository with the Ansible inventory file created in the previous chapter.

Go to the **openshift-install** directory and run the **git pull** command.

```
[root@console ~]# cd openshift-install  
[root@console openshift-install]# git pull
```

- 1.3. Inspect the files from the cloned repository.

Go to the **openshift-install/lab.example.com** directory and list the files.

```
[root@console openshift-install]# cd lab.example.com/
```

```
[root@console lab.example.com]# ls  
hosts-ucf README.md rootCA.crt rootCA.csr rootCA.key
```

Open the **hosts-ucf** file with a text editor.

- 1.4. Ensure that the following host groups are defined in the **[OSEv3:children]** group.

- masters
- etcd
- nodes
- nfs
- lb

- 1.5. Ensure that each host group is defined and contains nodes.

```
[masters]  
master1.lab.example.com openshift_ip=172.25.250.11  
  openshift_public_ip=172.25.250.11  
master2.lab.example.com openshift_ip=172.25.250.12  
  openshift_public_ip=172.25.250.12  
master3.lab.example.com openshift_ip=172.25.250.13  
  openshift_public_ip=172.25.250.13  
  
[etcd]  
master[1:3].lab.example.com  
  
[nodes]  
master1.lab.example.com openshift_node_labels="{'region': 'master'}" \  
  openshift_schedulable=false openshift_ip=172.25.250.11  
  openshift_public_ip=172.25.250.11  
master2.lab.example.com openshift_node_labels="{'region': 'master'}" \  
  openshift_schedulable=false openshift_ip=172.25.250.12  
  openshift_public_ip=172.25.250.12  
master3.lab.example.com openshift_node_labels="{'region': 'master'}" \  
  openshift_schedulable=false openshift_ip=172.25.250.13  
  openshift_public_ip=172.25.250.13  
infra1.lab.example.com openshift_node_labels="{'region': 'infra'}" \  
  openshift_ip=172.25.250.21 openshift_public_ip=172.25.250.21  
infra2.lab.example.com openshift_node_labels="{'region': 'infra'}" \  
  openshift_ip=172.25.250.22 openshift_public_ip=172.25.250.22  
infra3.lab.example.com openshift_node_labels="{'region': 'infra'}" \  
  openshift_ip=172.25.250.23 openshift_public_ip=172.25.250.23  
apps1.lab.example.com openshift_node_labels="{'region': 'primary', 'zone': 'local'}" \  
  openshift_ip=172.25.250.31 openshift_public_ip=172.25.250.31  
apps2.lab.example.com openshift_node_labels="{'region': 'primary', 'zone': 'local'}" \  
  openshift_ip=172.25.250.32 openshift_public_ip=172.25.250.32  
  
[nfs]  
console.lab.example.com  
  
[lb]  
console.lab.example.com containerized=false
```

2. From **workstation**, open a new terminal and run the **lab advanced-installation grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab advanced-installation grade
...
Comparing Entries in [OSEv3:vars]

. Checking openshift_deployment_type..... PASS
. Checking deployment_type..... PASS
. Checking openshift_release..... PASS
. Checking osm_cluster_network_cidr..... PASS
. Checking openshift_portal_net..... PASS
...
```

3. Run the playbook.

- 3.1. On the **console** VM, use the **ansible-playbook** command against the inventory file. The playbook is located at **/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml**.

Alternatively, use the Ansible inventory file located in the **/root/D0380/solutions/cluster-install** directory.



Note

The installation process takes approximately 30 to 45 minutes. Make sure that there are no errors during the execution of the playbook. Fatal errors display in red.



Note

You may disregard the Ansible Playbook following output warnings:

```
[DEPRECATION WARNING]: 'include' for playbook includes. You should use
'import_playbook' instead. This feature will be removed in version
2.8. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
[DEPRECATION WARNING]: The use of 'include' for tasks has
been deprecated. Use 'import_tasks' for static inclusions or
'include_tasks' for dynamic inclusions. This feature will be removed
in a future release.
Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
[DEPRECATION WARNING]: include is kept for backwards compatibility
but usage is discouraged. The module documentation details page may
explain more about this rationale.. This feature will be removed in a
future release. Deprecation warnings can be disabled by setting
deprecation_warnings=False in ansible.cfg.
[DEPRECATION WARNING]: The use of 'static' has been deprecated.
Use 'import_tasks' for static inclusion, or 'include_tasks' for
dynamic inclusion. This feature will be removed in a future release.
Deprecation
warnings can be disabled by setting deprecation_warnings=False in
ansible.cfg.
```

Ensure that the **PLAY RECAP** section does not display any **failed** tasks.

```
[root@console lab.example.com]# ansible-playbook -i hosts-ucf \
/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml
PLAY [Create initial host groups for localhost] ****
TASK [include_vars] ****
ok: [localhost]
PLAY [Populate config host groups] ****
TASK [Evaluate groups - g_etcd_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_master_hosts or g_new_master_hosts required] ***
skipping: [localhost]
TASK [Evaluate groups - g_node_hosts or g_new_node_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_lb_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_nfs_hosts required] ****
skipping: [localhost]
TASK [Evaluate groups - g_nfs_hosts is single host] ****
skipping: [localhost]
TASK [Evaluate groups - g_glusterfs_hosts required] ****
skipping: [localhost]

...output omitted...

PLAY RECAP ****
apps1.lab.example.com      : ok=252  changed=60    unreachable=0    failed=0
apps2.lab.example.com      : ok=252  changed=60    unreachable=0    failed=0
console.lab.example.com    : ok=126  changed=22    unreachable=0    failed=0
infra1.lab.example.com    : ok=252  changed=60    unreachable=0    failed=0
infra2.lab.example.com    : ok=252  changed=60    unreachable=0    failed=0
infra3.lab.example.com    : ok=252  changed=60    unreachable=0    failed=0
localhost                  : ok=14   changed=0     unreachable=0    failed=0
master1.lab.example.com   : ok=728  changed=191   unreachable=0    failed=0
```

```
master2.lab.example.com : ok=470  changed=127  unreachable=0  failed=0
master3.lab.example.com : ok=470  changed=127  unreachable=0  failed=0
```

4. Review the installation.

- 4.1. Run the Ansible **shell** ad hoc command on the **console** VM against the **nodes** group to ensure that the **atomic-openshift-node.service** service is running.

Success for this command indicates that the nodes were properly configured by the playbook.



Note

You can also run the script located at **/root/D0380/labs/cluster-install/ad-hoc.sh**.

```
[root@console lab.example.com]# ansible -i hosts-ucf nodes \
-m shell -a "systemctl is-active atomic-openshift-node.service"
master1.lab.example.com | SUCCESS | rc=0 >>
active

master3.lab.example.com | SUCCESS | rc=0 >>
active

infra1.lab.example.com | SUCCESS | rc=0 >>
active

infra2.lab.example.com | SUCCESS | rc=0 >>
active

master2.lab.example.com | SUCCESS | rc=0 >>
active

infra3.lab.example.com | SUCCESS | rc=0 >>
active

apps1.lab.example.com | SUCCESS | rc=0 >>
active

apps2.lab.example.com | SUCCESS | rc=0 >>
active
```

- 4.2. Connect to the cluster by logging in to the **master1** server.

```
[root@console lab.example.com]# ssh root@master1
[root@master1 ~]# oc whoami
system:admin
```

5. List the certificates.

Go to the **/etc/origin/master** directory and list the certificate keys.

```
[root@master1 ~]# cd /etc/origin/master
[root@master1 master]# ls -al *key
-rw-----. 3 root root 1679 Nov 11 14:33 admin.key
```

```
-rw-r--r--. 3 root root 1679 Nov 11 14:33 ca.key
-rw-----. 1 root root 1679 Nov 11 14:33 etcd.server.key
-rw-----. 1 root root 1708 Nov 11 14:34 master.etcd-client.key
-rw-----. 3 root root 1675 Nov 11 14:33 master.kubelet-client.key
-rw-----. 3 root root 1679 Nov 11 14:33 master.proxy-client.key
-rw-----. 1 root root 1679 Nov 11 14:33 master.server.key
-rw-----. 1 root root 1679 Nov 11 14:33 openshift-master.key
-rw-----. 1 root root 1679 Nov 11 14:51 openshift-router.key
-rw-----. 1 root root 1679 Nov 11 14:54 registry.key
-rw-----. 3 root root 1679 Nov 11 14:33 serviceaccounts.private.key
-rw-----. 3 root root 451 Nov 11 14:33 serviceaccounts.public.key
-rw-----. 3 root root 1675 Nov 11 14:33 service-signer.key
```

List the certificate files.

```
[root@master1 master]# ls -al *crt
-rw-r--r--. 3 root root 1216 Nov 11 14:33 admin.crt
-rw-r--r--. 3 root root 1220 Nov 11 14:33 ca-bundle.crt
-rw-r--r--. 3 root root 1220 Nov 11 14:33 ca.crt
-rw-r--r--. 1 root root 2794 Nov 11 14:33 etcd.server.crt
-rw-----. 1 root root 1895 Nov 11 14:30 master.etcd-ca.crt
-rw-----. 1 root root 5908 Nov 11 14:34 master.etcd-client.crt
-rw-r--r--. 3 root root 1233 Nov 11 14:33 master.kubelet-client.crt
-rw-r--r--. 3 root root 1184 Nov 11 14:33 master.proxy-client.crt
-rw-r--r--. 1 root root 2794 Nov 11 14:33 master.server.crt
-rw-r--r--. 1 root root 1265 Nov 11 14:33 openshift-master.crt
-rw-r--r--. 1 root root 2485 Nov 11 14:51 openshift-router.crt
-rw-r--r--. 1 root root 2607 Nov 11 14:54 registry.crt
-rw-r--r--. 3 root root 1115 Nov 11 14:33 service-signer.crt
```

6. Locate the following sections, which define the security certificates.

Use a text editor to open the file **/etc/origin/master/master-config.yaml**.

6.1. The **assetConfig** group defines the certificate to use for the web console:

```
assetConfig:
  ...
  servingInfo:
    bindAddress: 0.0.0.0:443
    bindNetwork: tcp4
    certFile: master.server.crt
```

6.2. The **controllerConfig** group defines the certificate to use for the controllers:

```
controllerConfig:
  serviceServingCert:
    signer:
      certFile: service-signer.crt
      keyFile: service-signer.key
```

6.3. The **etcdClientInfo** group defines the certificate for Etcd:

```
etcdClientInfo:
  ca: master.etcd-ca.crt
  certFile: master.etcd-client.crt
  keyFile: master.etcd-client.key
```

- 6.4. The **kubeletClientInfo** group defines the certificate for the nodes.

```
kubeletClientInfo:  
  ca: ca-bundle.crt  
  certFile: master.kubelet-client.crt  
  keyFile: master.kubelet-client.key
```

- 6.5. The **kubernetesMasterConfig** group defines the certificate for the orchestration service.

```
kubernetesMasterConfig:  
  ...  
  proxyClientInfo:  
    certFile: master.proxy-client.crt  
    keyFile: master.proxy-client.key
```

- 6.6. The **oauthConfig** group defines the certificate for the authentication:

```
oauthConfig:  
  ...  
  masterCA: ca-bundle.crt
```

- 6.7. The **serviceAccountConfig** group defines the certificate for the accounts token generation:

```
serviceAccountConfig:  
  ...  
  masterCA: ca-bundle.crt
```

- 6.8. The **servingInfo** group defines the certificate for the web console:

```
servingInfo:  
  bindAddress: 0.0.0.0:443  
  bindNetwork: tcp4  
  certFile: master.server.crt  
  clientCA: ca-bundle.crt
```

7. Inspect the certificate files from an application node.

- 7.1. Connect to the **apps1** node as the **root** user. When prompted, use **redhat** as the password.

```
[root@master1 master]# ssh root@apps1  
root@apps1's password: redhat
```

- 7.2. Inspect the certificate files stored on the node.

Go to the **/etc/origin/node** directory and list the files in the directory.

```
[root@apps1 ~]# cd /etc/origin/node  
[root@apps1 node]# ls -al
```

```
total 40
drwx----- 2 root root 266 Nov 20 16:24 .
drwxr-xr-x 3 root root 18 Nov 20 16:21 ..
-rw-r--r-- 1 root root 1220 Nov 20 16:21 ca.crt
-rw----- 1 root root 1253 Nov 20 16:24 node-config.yaml
-rw-r--r-- 1 root root 63 Nov 20 16:22 node-dnsMasq.conf
-rw----- 1 root root 26 Nov 20 16:23 resolv.conf
-rw-r--r-- 1 root root 2469 Nov 20 16:21 server.crt
-rw----- 1 root root 1679 Nov 20 16:21 server.key
-rw-r--r-- 1 root root 1233 Nov 20 16:21 system:node:apps1.lab.example.com.crt
-rw----- 1 root root 1675 Nov 20 16:21 system:node:apps1.lab.example.com.key
-rw----- 1 root root 6149 Nov 20 16:21
system:node:apps1.lab.example.com.kubeconfig
```

- 7.3. Use a text editor to open the **/etc/origin/node/node-config.yaml** configuration file:

Locate the **servingInfo** section, which defines the security certificate for the web console.

```
...
servingInfo:
  bindAddress: 0.0.0.0:10250
  certFile: server.crt
  clientCA: ca.crt
  keyFile: server.key
```

- 7.4. Log out of **apps1**.

```
[root@apps1 node]# exit
```

8. On the **master1** VM, run the **oc adm policy** command to enable administrative users to connect to the cluster from an external network.



Note

You can also run the script located at **/root/D0380/labs/cluster-install/set-privileges.sh** in the **console** VM.

```
[root@master1 master]# oc adm policy add-cluster-role-to-user \
cluster-admin admin
cluster role "cluster-admin" added: "admin"
```

9. Log out of the **master1** and **console** VMs.

```
[root@master1 master]# exit
[root@console lab.example.com]# exit
[student@workstation ~]$
```

10. Ensure that the **admin** user can access the installed cluster.

On the **workstation** VM, open a new terminal and run the **oc login** command.

```
[student@workstation ~]$ oc login https://console.lab.example.com \
-u admin -p redhat
The server uses a certificate an invalid certificate x509: certificate is not
authorized to sign other certificates
You can bypass the certificate check, but any data you send to the server could be
intercepted by others.

Use insecure connections? (y/n): y

Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

* default
  kube-public
  kube-system
  logging
  management-infra
  openshift
  openshift-infra
```

This concludes the guided exercise.

Lab: Building a Highly Available Cluster

In this lab, you will review the installation of OpenShift Container Platform by running an Ansible Playbook. You will also test the cluster by creating a project and an application.

Outcomes

You should be able to:

- Review the Ansible inventory file that defines the nodes for the OpenShift cluster.
- Ensure that all nodes are running in the cluster and that they are labeled.
- Ensure that the registry and the router pods are running in the cluster.
- Reach the OpenShift Container Platform web console from the **workstation** VM.
- Run the Ansible Playbook that performs post-installation tasks.
- Create an OpenShift project.
- Create an OpenShift application and expose it with an OpenShift service.

Before you begin

The guided exercises *Executing Pre-installation Tasks* and *Using the Advanced Installation Method* must have been completed. If not, complete the guided exercises before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab cluster-review setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project. It also installs the required files for this exercise.

```
[student@workstation ~]$ lab cluster-review setup
```

Steps

1. Ensure that labels are defined for the servers in the **nodes** group.

The **nodes** section in the **/home/student/D0380/labs/cluster-review/hosts-ucf** Ansible inventory file from the **workstation** VM defines the nodes.

2. Ensure that the following nodes are listed as schedulable in the cluster.
 - **apps1** and **apps2**
 - **infra1**, **infra2**, and **infra3**
3. Inspect the labels for each schedulable node.
4. Ensure that the router and registry pods are deployed in the **default** project.
 - 4.1. Ensure that there are three router pods in the **default** project and that they are distributed over the three infrastructure nodes.

- 4.2. Ensure that there are three registry pods in the **default** project and that they are distributed over the three infrastructure nodes.
- 4.3. Log out of the **master1** VM.
5. Ensure that the OpenShift web console is accessible from the workstation VM. Log in to the web console using **developer** as the user name and **redhat** as the password.
6. On the **workstation** VM, inspect the **/home/student/D0380/labs/cluster-review/post_install.yml** Ansible Playbook file and the modules it contains.
7. Run the evaluated Ansible Playbook.
8. Ensure that the Red Hat Container Registry from the **registry.lab.example.com:5000** host has the **hello-openshift** image.
9. Create an application using the **hello-openshift** image.
10. Expose the **cluster-app** application with a host name of **cluster.apps.lab.example.com**.
 - 10.1. Expose the application.
 - 10.2. Confirm that the application resolves to the host name.
 - 10.3. Go to the **student**'s home directory.

```
[student@workstation cluster-review ]$ cd
```
11. Run the **lab cluster-review grade** script to grade your work.

```
[student@workstation ~]$ lab cluster-review grade
```
12. Clean up the lab.

Delete the **cluster-install** project.

Solution

In this lab, you will review the installation of OpenShift Container Platform by running an Ansible Playbook. You will also test the cluster by creating a project and an application.

Outcomes

You should be able to:

- Review the Ansible inventory file that defines the nodes for the OpenShift cluster.
- Ensure that all nodes are running in the cluster and that they are labeled.
- Ensure that the registry and the router pods are running in the cluster.
- Reach the OpenShift Container Platform web console from the **workstation** VM.
- Run the Ansible Playbook that performs post-installation tasks.
- Create an OpenShift project.
- Create an OpenShift application and expose it with an OpenShift service.

Before you begin

The guided exercises *Executing Pre-installation Tasks* and *Using the Advanced Installation Method* must have been completed. If not, complete the guided exercises before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab cluster-review setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project. It also installs the required files for this exercise.

```
[student@workstation ~]$ lab cluster-review setup
```

Steps

1. Ensure that labels are defined for the servers in the **nodes** group.

The **nodes** section in the **/home/student/D0380/labs/cluster-review/hosts-ucf** Ansible inventory file from the **workstation** VM defines the nodes.

Review the labels defined for the nodes.

```
[student@workstation ~]$ cat /home/student/D0380/labs/cluster-review/hosts-ucf
...
[nodes]
master1.lab.example.com openshift_node_labels="{'region': 'master'}" ...
master2.lab.example.com openshift_node_labels="{'region': 'master'}" ...
master3.lab.example.com openshift_node_labels="{'region': 'master'}" ...
infra1.lab.example.com openshift_node_labels="{'region': 'infra'}" ...
infra2.lab.example.com openshift_node_labels="{'region': 'infra'}" ...
infra3.lab.example.com openshift_node_labels="{'region': 'infra'}" ...
apps1.lab.example.com openshift_node_labels="{'region': 'primary', 'zone': 'local'}"
...
apps2.lab.example.com openshift_node_labels="{'region': 'primary', 'zone': 'local'}"
...
```

2. Ensure that the following nodes are listed as schedulable in the cluster.

- **apps1** and **apps2**
- **infra1**, **infra2**, and **infra3**

2.1. Connect to the **master1** VM from the **workstation** VM.

Use the **ssh** command to log in to the **master1** VM as the **root** user

```
[student@workstation ~]$ ssh root@master1
[root@master1 ~]#
```

2.2. Ensure that you are logged in as the **system:admin** user.

Run the **oc whoami** command.

```
[root@master1 ~]# oc whoami
system:admin
```

2.3. Ensure that the following nodes are listed and are schedulable.

- **apps1** and **apps2**
- **infra1**, **infra2**, and **infra3**

Run the **oc get nodes** command to list all the nodes in the cluster.

```
[root@master1 ~]# oc get nodes
NAME          STATUS    AGE      VERSION
apps1.lab.example.com   Ready    6d       v1.6.1+5115d708d7
apps2.lab.example.com   Ready    6d       v1.6.1+5115d708d7
infra1.lab.example.com  Ready    6d       v1.6.1+5115d708d7
infra2.lab.example.com  Ready    6d       v1.6.1+5115d708d7
infra3.lab.example.com  Ready    6d       v1.6.1+5115d708d7
master1.lab.example.com Ready, SchedulingDisabled 6d       v1.6.1+5115d708d7
master2.lab.example.com Ready, SchedulingDisabled 6d       v1.6.1+5115d708d7
master3.lab.example.com Ready, SchedulingDisabled 6d       v1.6.1+5115d708d7
```

The absence of the **SchedulingDisabled** in the status column indicates the nodes are schedulable.

3. Inspect the labels for each schedulable node.

Use the **oc describe NODE** command to retrieve the labels for each schedulable node. Repeat the following command for the **apps2**, **infra1**, **infra2**, and **infra3** nodes.

The following table lists the expected labels.

Nodes	Label
apps1 and apps2	region=primary, zone=local
infra1 , infra2 , and infra3	region=infra

```
[root@master1 ~]# oc describe node apps1.lab.example.com | grep Labels -A 4
```

```
Labels: <beta.kubernetes.io/arch=amd64
       beta.kubernetes.io/os=linux
       kubernetes.io/hostname=apps1.lab.example.com
       region=primary
       zone=local
```

4. Ensure that the router and registry pods are deployed in the **default** project.
 - 4.1. Ensure that there are three router pods in the **default** project and that they are distributed over the three infrastructure nodes.

On the **master1** VM, run the **oc get pods** against the **default** project. The output lists three routers, each running on a different node.

```
[root@master1 ~]# oc get pods -o wide -n default | grep router
router-1-b685c 1/1 Running 0 6d 172.25.250.21 infra1.lab.example.com
router-1-f1lwk 1/1 Running 0 6d 172.25.250.23 infra3.lab.example.com
router-1-fhlgj 1/1 Running 0 6d 172.25.250.22 infra2.lab.example.com
```
 - 4.2. Ensure that there are three registry pods in the **default** project and that they are distributed over the three infrastructure nodes.
- On the **master1** VM, run the **oc get pods** against the **default** project. The output lists three routers, each running in a different node.

```
[root@master1 ~]# oc get pods -o wide -n default | grep docker-registry
docker-registry-1-30g4q 1/1 Running 0 6d 10.128.0.2 infra1.lab.example.com
docker-registry-1-5rlkr 1/1 Running 0 6d 10.130.0.3 infra2.lab.example.com
docker-registry-1-g4tpg 1/1 Running 0 6d 10.129.2.3 infra3.lab.example.com
```

- 4.3. Log out of the **master1** VM.
- ```
[root@master1 ~]# exit
[student@workstation ~]$
```
5. Ensure that the OpenShift web console is accessible from the workstation VM. Log in to the web console using **developer** as the user name and **redhat** as the password.
- On the **workstation** VM, open Firefox and access **https://console.lab.example.com:443**. Accept the self-signed security certificate.
- Use **developer** as the user name and **redhat** as the password. The page should read **Welcome to OpenShift**.
6. On the **workstation** VM, inspect the **/home/student/D0380/labs/cluster-review/post\_install.yml** Ansible Playbook file and the modules it contains.

The lab script downloaded the **/home/student/D0380/labs/cluster-review/post\_install.yml** Ansible Playbook file, which is responsible for performing the post-installation tasks.

- 6.1. Inspect the variables used by the playbook.

The first block defines the variables used by the playbook.

```
vars:
 master: master1.lab.example.com
```

## 6.2. Inspect the task that verifies that the master node is running.

The first task ensures that the **atomic-openshift-master-api** and **atomic-openshift-master-controllers** services are started and enabled on the master nodes.

```
- name: Check for OCP Services
 service:
 name: "{{ item }}"
 state: started
 enabled: true
 when: inventory_hostname in groups['masters']
 with_items:
 - atomic-openshift-master-controllers
 - atomic-openshift-master-api
```

## 6.3. Inspect the task that configures the package exclusions.

Inspect the next task, which adds entries to the **exclude** directive in each host's **/etc/yum.conf** file.

```
- name: Re-add OpenShift package exclusions
 command: "/usr/sbin/atomic-openshift-excluder exclude"
```

## 6.4. Inspect the tasks that update the registry console to use an internal registry.

The registry console pod on the master nodes is misconfigured after the default installation. The registry configuration must be updated to use an internal registry to support offline environments. The module execution is limited to the **masters** group in the Ansible inventory file.

```
- name: Fix registry console
 command: "oc set image --source=docker dc/registry-console registry-console=registry.lab.example.com:5000/openshift3/registry-console:v3.6"
 when: inventory_hostname in groups['masters']

- name: Wait for registry-console to re-deploy
 pause:
 seconds: 15
 prompt: "Waiting for the registry-console to re-deploy"
```

## 6.5. Inspect the image stream configuration resources in the cluster.

These resources must be updated because they download the S2I images from the Red Hat image registry, but the classroom does not have Internet access.

```
- name: Edit RHEL7 Image Streams
 copy:
 src: files/image-streams-rhel7.json
 dest: /usr/share/openshift/examples/image-streams/image-streams-rhel7.json
 register: rhel7_is_result
```

```

when: "master in inventory_hostname"

- name: delete_openshift_is
 command: "/usr/bin/oc delete is -n openshift --all"
 when: rhel7_is_result.changed and "master in inventory_hostname"

- name: create_rhel7_is
 command: "/usr/bin/oc create -f /usr/share/openshift/examples/image-streams/
image-streams-rhel7.json -n openshift"
 when: rhel7_is_result.changed and "master in inventory_hostname"

```

To make the changes to the installed **ImageStream** resources, a JSON file is provided in the **/home/student/D0380/labs/cluster-review/files** directory. Open the **image-streams-rhel7.json** file in that directory, and ensure that the **from** attribute for all the image streams point to the **registry.lab.example.com:5000** private registry instead of **registry.access.redhat.com**.

```

...
 "from": {
 "kind": "DockerImage",
 "name": "registry.lab.example.com:5000/openshift3/ruby-20-rhel7:latest"
 },
...

```

#### 6.6. Inspect the tasks that reload the changes made so far.

Inspect the tasks that are responsible for restarting the OpenShift master service.

```

- name: Restart OpenShift Master service to apply authentication changes
 service:
 name: "{{ item }}"
 state: restarted
 with_items:
 - atomic-openshift-master-controllers
 - atomic-openshift-master-api
 when: inventory_hostname in groups['masters']

- name: Waiting for the master to restart
 pause:
 seconds: 10
 prompt: "Waiting for the master to settle"

```

#### 7. Run the evaluated Ansible Playbook.

On the **workstation** VM, run the Ansible Playbook **/home/student/D0380/labs/cluster-review/post\_install.yml** using the **hosts-ucf** inventory file in the same directory. Ensure that the **RECAP** section does not contain any failed tasks.

##### 7.1. Navigate to the directory that contains the playbook.

On the **workstation** VM, open a new terminal and go to the **/home/student/D0380/labs/cluster-review/** directory.

```
[student@workstation ~]$ cd D0380/labs/cluster-review
[student@workstation cluster-review]$
```

## 7.2. Run the post-installation playbook.

Run the **post\_install.yml** Ansible Playbook with the **hosts-ucf** file as the inventory file.

```
[student@workstation cluster-review]$ ansible-playbook -i hosts-ucf \
post_install.yml
```

Upon execution, ensure that the **RECAP** section does not contain any failed tasks.

```
PLAY RECAP ****
apps1.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
apps2.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
infra1.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
infra2.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
infra3.lab.example.com : ok=2 changed=1 unreachable=0 failed=0
master1.lab.example.com : ok=10 changed=6 unreachable=0 failed=0
master2.lab.example.com : ok=5 changed=3 unreachable=0 failed=0
master3.lab.example.com : ok=5 changed=3 unreachable=0 failed=0
```



### Note

The counts in the play summary may be different on your system.

8. Ensure that the Red Hat Container Registry from the **registry.lab.example.com:5000** host has the **hello-openshift** image.

On the **workstation** VM, use the **docker-registry-cli** command to browse the Red Hat Container Registry at **registry.lab.example.com** on port 5000. Use the **search** option to browse images in the registry.

```
[student@workstation cluster-review]$ docker-registry-cli \
registry.lab.example.com:5000 search hello-openshift

1) Name: openshift/hello-openshift
Tags: latest

1 images found !
```

9. Create an application using the **hello-openshift** image.

- 9.1. On the **workstation** VM, create a new project called **cluster-install** as the **developer** user with a password of **redhat**.

On the **workstation** VM, run the **oc login** command to log in to the cluster.

```
[student@workstation cluster-review]$ oc login \
https://console.lab.example.com:443 \
-u developer -p redhat
Login successful.

You don't have any projects. You can try to create a new project, by running
```

```
oc new-project <projectname>
```

## 9.2. Create the **cluster-install** project.

On the **workstation** VM, run the **oc new-project** command to create the project.

```
[student@workstation cluster-review]$ oc new-project cluster-install
Now using project "cluster-install" on server "https://
console.lab.example.com:443".
...
```

## 9.3. Create the **cluster-app** application using the **hello-openshift** image as the base image and configure the offline registry as not secure.

Use the **--insecure-registry** option, because the offline registry is not secure.

Optionally, run the **create-app.sh** script available at **/home/student/D0380/solutions/cluster-review**.

```
[student@workstation cluster-review]$ oc new-app --name=cluster-app \
--docker-image=registry.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry
--> Found Docker image ddaec72 (23 minutes old) from
 registry.lab.example.com:5000 for "registry.lab.example.com:5000/openshift/
 hello-openshift"
...
--> Creating resources ...
 imagestream "cluster-app" created
 deploymentconfig "cluster-app" created
 service "cluster-app" created
--> Success
Run 'oc status' to view your app.
```

## 9.4. Review the status of the application deployment.

Run the **oc get pods** command until one pod named **cluster-app-1-hpzn3** shows with a status of **Running** and is marked as ready (**1/1**).

```
[student@workstation cluster-review]$ oc get pods
NAME READY STATUS RESTARTS AGE
cluster-app-1-hpzn3 1/1 Running 0 10m
```

## 10. Expose the **cluster-app** application with a host name of **cluster.apps.lab.example.com**.

### 10.1. Expose the application.

Use the **oc expose svc APP NAME** command.

```
[student@workstation cluster-review]$ oc expose svc cluster-app \
--hostname=cluster.apps.lab.example.com
route "cluster-app" exposed
```

10.2. Confirm that the application resolves to the host name.

Run the **curl** command against the URL.

```
[student@workstation cluster-review]$ curl http://cluster.apps.lab.example.com
Hello OpenShift!
```

10.3. Go to the **student**'s home directory.

```
[student@workstation cluster-review]$ cd
```

11. Run the **lab cluster-review grade** script to grade your work.

```
[student@workstation ~]$ lab cluster-review grade
```

12. Clean up the lab.

Delete the **cluster-install** project.

```
[student@workstation ~]$ oc delete project cluster-install
```

# Summary

In this chapter, you learned:

- There are often three options available for deploying OpenShift Container Platform: bare metal, virtualization solutions, and cloud computing solutions. All these options have their benefits, the decisive factors being deployment and maintenance costs, administrator training, and ROI.
- You can deploy and configure their environment in various fashions:
  - Single master and multiple nodes
  - Single master
  - Multiple Etcd servers, and multiple nodes
  - Multiple masters with OpenShift native HA
  - Stand-alone registry server, with a single master or multiple master nodes
- You must configure Red Hat Container Registry on all master and node hosts before running the installer. OpenShift Container Platform uses Red Hat Container Registry to store containers and images, which provides the benefit of configuring the Docker storage options before installing OpenShift Container Platform.
- You can customize the installation of OpenShift Container Platform with the Ansible advanced installation method. In contrast to the quick installation method, the advanced installation method is not interactive. Instead, it uses a set of prepopulated configuration options to determine how to install the OpenShift cluster.





## CHAPTER 5

# PROVISIONING PERSISTENT STORAGE

| Overview          |                                                                                                                                                                                                                                                                                                              |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Goal</b>       | Describe storage providers, configure a provider, create a storage class, and test the configuration.                                                                                                                                                                                                        |
| <b>Objectives</b> | <ul style="list-style-type: none"><li>• Describe the various storage providers available and the concept of storage classes.</li><li>• Configure storage providers using the advanced installation method.</li><li>• Configure and test the Red Hat Gluster Storage CNS provider in an HA cluster.</li></ul> |
| <b>Sections</b>   | <ul style="list-style-type: none"><li>• Describing Storage Providers (and Quiz)</li><li>• Configuring Storage Providers (and Guided Exercise)</li><li>• Configuring and Testing Red Hat Gluster Storage Container-Native Storage (and Guided Exercise)</li></ul>                                             |
| <b>Lab</b>        | Provisioning Persistent Storage                                                                                                                                                                                                                                                                              |

# Describing Storage Providers

## Objective

After completing this section, students should be able to describe the various storage providers available and the concept of storage classes.

## Persistent Storage

OpenShift Container Platform uses the Kubernetes *persistent volume (PV)* framework to allow administrators to provision persistent storage for a cluster. OpenShift users can define *persistent volume claims (PVCs)* to request PV resources without having specific knowledge of the underlying storage infrastructure.

PVs are defined by a *PersistentVolume* API object, which represents a piece of existing networked storage in the cluster that has been provisioned by an administrator.

PVCs are defined by a *PersistentVolumeClaim* API object, which represents a definition request for storage.

## Persistent Volume Types

OpenShift Container Platform supports the following persistent volume types:

- NFS
- HostPath
- Red Hat Gluster Storage
- OpenStack Cinder
- Ceph RBD
- AWS Elastic Block Store (EBS)
- GCE Persistent Disk
- iSCSI
- Fibre Channel
- Azure Disk
- Azure File
- VMware vSphere

For example, a Red Hat Gluster Storage persistent volume is configured as follows:

```
apiVersion: v1
kind: PersistentVolume
metadata:
 name: gluster-pv ①
spec:
```

```

capacity:
 storage: 5Gi 2
 accessModes:
 - ReadWriteMany 3
 glusterfs: 4
 endpoints: glusterfs-cluster 5
 path: glustervol 6
 readOnly: false
 persistentVolumeReclaimPolicy: Recycle 7

```

- 1** The name of the PV, which is referenced in pod definitions or displayed in various **oc volume** commands.
- 2** The amount of storage allocated to this volume.
- 3** Used as labels to match a PV and a PVC.
- 4** Defines the volume type being used. In this case, the **glusterfs** plug-in is defined.
- 5** The name of the endpoints that defines the Red Hat Gluster Storage cluster.
- 6** The volume that is accessed by Red Hat Gluster Storage, as shown in the **gluster volume status** command.
- 7** The **Recycle** volume reclaim policy indicates that the volume will be preserved after the pods accessing it are stopped. For Red Hat Gluster Storage, the accepted values include **Retain** and **Delete**.



## Note

For further configuration examples, refer to the *Configuring Persistent Storage* guide at [https://docs.openshift.org/latest/install\\_config/persistent\\_storage/index.html](https://docs.openshift.org/latest/install_config/persistent_storage/index.html)

### Access Modes

You can mount a persistent volume on a host in any way supported by the resource provider. Providers will have different capabilities and each PV's access modes are set to the specific modes supported by that particular volume. For example, NFS can support multiple read and write clients, but a specific NFS PV might be exported on the server as read-only. Each PV has its own set of access modes describing that specific PV's capabilities.

| Volume Type         | ReadWriteOnce | ReadOnlyMany | ReadWriteMany |
|---------------------|---------------|--------------|---------------|
| AWS EBS             | Yes           | No           | No            |
| Azure Disk          | Yes           | No           | No            |
| Ceph RBD            | Yes           | Yes          | No            |
| Fibre Channel       | Yes           | Yes          | No            |
| GCE Persistent Disk | Yes           | No           | No            |
| GlusterFS           | Yes           | Yes          | Yes           |
| HostPath            | Yes           | No           | No            |
| iSCSI               | Yes           | Yes          | No            |
| NFS                 | Yes           | Yes          | Yes           |
| OpenStack Cinder    | Yes           | No           | No            |

| Volume Type    | ReadWriteOnce | ReadOnlyMany | ReadWriteMany |
|----------------|---------------|--------------|---------------|
| VMware vSphere | Yes           | No           | No            |

## Storage Classes

Storage classes manage and enable persistent storage in OpenShift Container Platform.

Storage class resource objects describe and classify storage that can be requested by PVCs and provides a means for passing parameters for dynamically provisioned storage on demand. Storage class objects also serve as a management mechanism for controlling different levels of storage and access to the storage. Cluster administrators define and create the storage class objects that users request without the need for any intimate knowledge of the underlying storage volume sources.

OpenShift Container Platform uses Heketi for the management of dynamic storage. Heketi provides a RESTful management interface for the life cycle of GlusterFS volumes. With the integration of Heketi, OpenShift Container Platform can dynamically provision GlusterFS volumes, as the service automatically determines the location for bricks across the cluster. Heketi also ensures that the bricks and their replicas are spread across different failure domains.

The following example defines a Red Hat Gluster Storage class:

```
apiVersion: storage.k8s.io/v1
kind: StorageClass
metadata:
 name: gluster-heketi ①
spec:
 provisioner: kubernetes.io/glusterfs ②
 parameters:
 endpoint: "storage-endpoints" ③
 resturl: "http://127.0.0.1:8081" ④
 restuser: "admin" ⑤
 restuserkey: "Any Secret Key Value" ⑥
```

- ① Name of this storage class.
- ② Storage class provisioner.
- ③ The GlusterFS-defined endpoints as seen using the `oc get endpoints` command.
- ④ REST API service or Heketi service URL which provision GlusterFS volumes on demand. The general format must be `address:port` and this is a mandatory parameter for Red Hat Gluster Storage dynamic provisioner.
- ⑤ REST API service or Heketi user who has access to create volumes in the trusted storage pool.
- ⑥ REST user key. This can be any value.

### Configuring Persistent Volume Claims

PVCs can optionally request a specific storage class by specifying its name in the `storageClassName` attribute. Only PVs of the requested class, with the same storage class name as the PVC, can be bound to the PVC. You can configure dynamic provisioners to service one or more storage classes. Create a PV on demand that matches the specifications in the PVC.

For example, a Red Hat Gluster Storage persistent volume claim definition is configured as follows:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: gluster-pvc
spec:
 accessModes:
 - ReadWriteOnce
 storageClassName: gluster-heketi ①
 resources:
 requests:
 storage: 5Gi ②
```

- ①** The name that matches the storage class.
- ②** The amount of storage requested.

### Creating Persistent Volume Claims

The following steps describe how to create a persistent volume claim that uses a storage class.

1. Define the persistent volume claim in a YAML file.

Use a persistent volume claim metadata annotation to identify the storage class:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 annotations:
 volume.beta.kubernetes.io/storage-class: gluster-container
...

```

2. Create a persistent volume claim using the **oc create -f** command.

Cluster administrators can also set a default storage class for all PVCs. When a default storage class is configured, the PVC must explicitly ask for **StorageClass** or **storageClassName** annotations which are set to "" to get bound to a PV with no storage class.

### Setting the Default Storage Class

The following steps describe how to set or change a default storage class.

1. Use the **oc get storageclass** command to identify the existing storage classes.
2. Change the value of the **storageclass.kubernetes.io/is-default-class** annotation to false for the current default storage class.
3. Add or modify the **storageclass.kubernetes.io/is-default-class=true** annotation to set the default storage class.

## Dynamic Provisioning of Volumes Using Container-Native Storage (CNS)

Dynamic provisioning enables provisioning of a Red Hat Gluster Storage volume to a running application container without having to create the volume beforehand. The volume is created

dynamically as the claim request comes in, and a volume of exactly the same size is provisioned to the application containers.

#### Configuring Dynamic Provisioning of Volumes

To configure dynamic provisioning of volumes, the cluster administrator must define storage class objects that describe named classes of storage offered in a cluster. After creating a storage class, create a secret for Heketi authentication before proceeding with the creation of a persistent volume claim.

1. Define the storage class in a YAML file.

When configuring a storage class object for persistent volume provisioning, you must describe the type of provisioner to use, for example, **kubernetes.io/glusterfs**, and the parameters that will be used by the provisioner when it provisions a persistent volume belonging to the class.

Parameter examples include:

##### **resturl**

REST service or Heketi service URL which provisions Red Hat Gluster Storage volumes on demand. The general format must be **IP-address:port** and this is a mandatory parameter for the Red Hat Gluster Storage dynamic provisioner.

##### **restuser**

REST API service or Heketi user who has access to create volumes in the trusted storage pool.

##### **secretNamespace** and **secretName**

Together these parameters identify the secret instance that contains the user password that is used when communicating with the REST API service. If neither value is specified then an empty password is used.

2. Create the storage class using the **oc create -f** command.

## Use Cases for User-defined Storage Classes

The following use cases present user-defined storage classes and dynamic provisioning using Google Cloud Platform Compute Engine (GCE).

#### Basic Dynamic Provisioning with Two Types of Storage Classes

Use storage classes to differentiate and delineate storage levels and how they are used. In both of these cases, the **cluster-admin** role configures two distinct classes of storage in GCE by modifying the metadata in each of the storage class definitions.

##### Slow

Inexpensive, efficient, and optimized for sequential data operations, for example, slower reading and writing:

```
metadata:
 name: slow
```

##### Fast

Optimized for higher rates of random *input and output operations per second (IOPS)* and sustained throughput, for example, faster reading and writing:

```
metadata:
 name: fast
```

By creating these storage classes, the cluster administrator allows users to create claims requesting a particular level or service of storage class.

## Selector-Label Volume Binding

By implementing selectors and labels, users can target provisioned storage using identifiers defined by cluster administrators.

For statically provisioned storage, users seeking persistent storage must know several identifying attributes of a PV in order to deploy and bind a PVC. This creates several problematic situations. Users might have to contact a cluster administrator to either deploy the PVC or provide the PV values. PV attributes alone do not convey the intended use of the storage volumes, nor do they provide methods by which volumes can be grouped.

Selector and label attributes can be used to abstract PV details from the user while providing cluster administrators a way of identifying volumes with a descriptive and customizable tag. Using the selector-label method of binding, users are only required to know which labels are defined by the administrator.



### Note

The selector-label feature is currently only available for statically provisioned storage and is currently not implemented for dynamically provisioned storage.

## Defining a Persistent Volume and Persistent Volume Claim with Labels

As the **cluster-admin** user, define the PV. This example uses a GlusterFS volume.

- 1 Define the persistent volume with labels:

```
cat gluster-pv.yaml

apiVersion: v1
kind: PersistentVolume
metadata:
 name: gluster-volume
 labels: ①
 volume-type: ssd
 aws-availability-zone: us-east-1
spec:
 capacity:
 storage: 2Gi
 accessModes:
 - ReadWriteMany
 glusterfs:
 endpoints: glusterfs-cluster
 path: myVol1
 readOnly: false
 persistentVolumeReclaimPolicy: Recycle
```

- ① A PVC whose selectors match all of a PV's labels will be bound, assuming a PV is available.
2. Define the persistent volume claim with selectors:

```
cat gluster-pvc.yaml

apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: gluster-claim
spec:
 accessModes:
 - ReadWriteMany
 resources:
 requests:
 storage: 1Gi
 selector: ①
 matchLabels: ②
 volume-type: ssd
 aws-availability-zone: us-east-1
```

- ① Begin selectors section.
- ② List all labels by which the user is requesting storage. Must match all labels of the targeted PV.

## References

Further information is available in the chapter on Dynamic Provisioning and Creating Storage Classes in the *Installation and Configuration Guide* for OpenShift Container Platform 3.6; at

[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/3.6](https://access.redhat.com/documentation/en-us/openshift_container_platform/3.6)

# Quiz: Describing Storage Providers

Choose the correct answers to the following questions:

1. Which statement is true about persistent volume access modes?
  - a. Persistent volumes can only support one read and write client.
  - b. Each persistent volume access mode is set to the specific modes supported by that particular volume.
  - c. All providers have the same capabilities but are configured based on particular access modes.
  - d. All persistent volumes support at least two separate access modes when mounted on a host.
2. Which three statements describe characteristics of a **StorageClass** resource object? (Choose three.)
  - a. Provides intimate knowledge about underlying storage volume sources.
  - b. Allows for passing parameters for dynamically provisioned storage on demand.
  - c. Serves as a management mechanism for controlling different levels of storage.
  - d. Can be defined and created by any authenticated OpenShift cluster user.
  - e. Serves as a management mechanism for controlling access to the storage.
3. Which statement is true about dynamic provisioning using Container-Native Storage (CNS)?
  - a. Dynamic provisioning with CNS disables provisioning of a Red Hat Gluster Storage volume to a running application container without an existing volume.
  - b. Volumes are created dynamically as the claim request is made.
  - c. Volumes are randomly sized when provisioned to the application containers.
  - d. Named classes describe storage offered in a cluster after creation by any authenticated cluster user.
4. Which two selector-label volume binding configurations allow users to target and bind provisioned storage using identifiers? (Choose two.)
  - a. An available persistent volume.
  - b. Persistent volume claim selectors match all persistent volume labels.
  - c. Persistent volume claim selectors match at least one persistent volume label.
  - d. Storage must be implemented as storage that is provisioned dynamically.

## Solution

Choose the correct answers to the following questions:

1. Which statement is true about persistent volume access modes?
  - a. Persistent volumes can only support one read and write client.
  - b. **Each persistent volume access mode is set to the specific modes supported by that particular volume.**
  - c. All providers have the same capabilities but are configured based on particular access modes.
  - d. All persistent volumes support at least two separate access modes when mounted on a host.
2. Which three statements describe characteristics of a **StorageClass** resource object? (Choose three.)
  - a. Provides intimate knowledge about underlying storage volume sources.
  - b. **Allows for passing parameters for dynamically provisioned storage on demand.**
  - c. **Serves as a management mechanism for controlling different levels of storage.**
  - d. Can be defined and created by any authenticated OpenShift cluster user.
  - e. **Serves as a management mechanism for controlling access to the storage.**
3. Which statement is true about dynamic provisioning using Container-Native Storage (CNS)?
  - a. Dynamic provisioning with CNS disables provisioning of a Red Hat Gluster Storage volume to a running application container without an existing volume.
  - b. **Volumes are created dynamically as the claim request is made.**
  - c. Volumes are randomly sized when provisioned to the application containers.
  - d. Named classes describe storage offered in a cluster after creation by any authenticated cluster user.
4. Which two selector-label volume binding configurations allow users to target and bind provisioned storage using identifiers? (Choose two.)
  - a. **An available persistent volume.**
  - b. **Persistent volume claim selectors match all persistent volume labels.**
  - c. Persistent volume claim selectors match at least one persistent volume label.
  - d. Storage must be implemented as storage that is provisioned dynamically.

# Configuring Storage Providers

## Objectives

After completing this section, students should be able to configure storage providers using the advanced installation method.

## Configuring Red Hat Gluster Storage

Red Hat Gluster Storage is a software-only, scale-out storage solution that provides flexible and agile unstructured data storage. It can be installed on commodity servers and storage hardware or deployed to the public cloud using Red Hat Gluster Storage Server for Public Cloud with Amazon Web Services (AWS), Microsoft Azure, or Google Cloud.

Red Hat Gluster Storage is a key building block that aggregates various storage servers over different network interfaces and connects them to form a single large parallel network file system.

### Requirements for Storage Nodes

To use containerized Red Hat Gluster Storage persistent storage:

- At least three storage nodes is required.
- Each storage node must have at least one raw block device with at least 100 GB available.

## Container-Native Storage (CNS)

This solution addresses the use case where applications require both shared file storage and the flexibility of a converged infrastructure with compute and storage instances that are scheduled and run from the same set of hardware.

### Natively Hosted Red Hat Gluster Storage Inventory File

The following example describes how to use a *bring your own (BYO)* host inventory for a cluster with natively hosted, containerized Red Hat Gluster Storage for applications. A storage class is automatically created when you use this inventory file.

- In the inventory file, add **glusterfs** to the **[OSEv3:children]** section to enable the **[glusterfs]** group:

```
[OSEv3:children]
masters
nodes
glusterfs ①
```

- In the **[OSEv3:vars]** section of the inventory file, add the **osn\_storage\_plugin\_deps=['ceph', 'glusterfs']** variable to install the default **ceph** and **glusterfs** plug-in dependencies:

```
[OSEv3:vars]
...output omitted...
osn_storage_plugin_deps=['ceph', 'glusterfs']
```

- Add the **[glusterfs]** group, which contains the nodes that host the Red Hat Gluster Storage storage pods. At a minimum, each node must have a **glusterfs\_devices** variable defined. This variable is a list of block devices that the nodes have access to, and which are intended solely for Red Hat Gluster Storage. These block devices are formatted by the installer and must be bare; that is, have no data and not marked as LVM PVs:

```
[glusterfs] ①
node0 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]' ②
node1 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
node2 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
```

- ① Group name for Red Hat Gluster Storage pods  
② List of Gluster devices on each Red Hat Gluster Storage node
- The following is an example of a completed inventory file for natively hosted Red Hat Gluster Storage:

```
[OSEv3:children]
masters
nodes
glusterfs

[OSEv3:vars]
ansible_ssh_user=root
openshift_deployment_type=openshift-enterprise
osn_storage_plugin_deps=['ceph', 'glusterfs']

[masters]
master

[nodes]
master openshift_schedulable=False
node0 openshift_schedulable=True
node1 openshift_schedulable=True
node2 openshift_schedulable=True

[glusterfs]
node0 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
node1 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
node2 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
```

### Running the Playbook for Native Red Hat Gluster Storage

Use the following command format to run the playbook for a natively hosted Red Hat Gluster Storage deployment:

```
[user@demo playbooks]$ ansible-playbook -i inventory-file playbook-file
```

The playbook reads as follows:

```

- name: Open firewall ports for GlusterFS nodes
 hosts: glusterfs
 vars:
 os_firewall_allow:
 - service: glusterfs_sshd
```

```

 port: "2222/tcp"
 - service: glusterfs_daemon
 port: "24007/tcp"
 - service: glusterfs_management
 port: "24008/tcp"
 - service: glusterfs_bricks
 port: "49152-49251/tcp"
roles:
 - role: os_firewall
 when:
 - openshift_storage_glusterfs_is_native | default(True) | bool

 - name: Open firewall ports for GlusterFS registry nodes
 hosts: glusterfs_registry
 vars:
 os_firewall_allow:
 - service: glusterfs_sshd
 port: "2222/tcp"
 - service: glusterfs_daemon
 port: "24007/tcp"
 - service: glusterfs_management
 port: "24008/tcp"
 - service: glusterfs_bricks
 port: "49152-49251/tcp"
 roles:
 - role: os_firewall
 when:
 - openshift_storage_glusterfs_registry_is_native | default(True) | bool

 - name: Configure GlusterFS
 hosts: oo_first_master
 roles:
 - role: openshift_storage_glusterfs
 when: groups.oo_glusterfs_to_config | default([]) | count > 0

```

## External Container-Ready Storage (CRS) with Heketi

This solution addresses the use case where a dedicated Red Hat Gluster Storage cluster is available external to the OpenShift cluster, and storage is provisioned from the Red Hat Gluster Storage cluster. In this mode, Heketi also runs outside the cluster and can be collocated with a Red Hat Gluster Storage node.

Heketi provides a RESTful management interface which can be used to manage the life cycle of Red Hat Gluster Storage volumes.

### Externally Hosted Red Hat Gluster Storage Inventory File

The following example describes an installation for externally hosted Red Hat Gluster Storage. You can use your own inventory file. The playbook automatically creates a storage class.

In the inventory file, add **glusterfs** to the **[OSEv3:children]** section to enable the **[glusterfs]** group:

```

[OSEv3:children]
masters
nodes
glusterfs ①

```

① Specifies that Red Hat Gluster Storage nodes exist.

Add variables to the **[OSEv3:vars]** section to define the Red Hat Gluster Storage installation as external:

```
openshift_storage_glusterfs_is_native=False ①
openshift_storage_glusterfs_heketi_url=172.0.0.1 ②
osn_storage_plugin_deps=['ceph', 'glusterfs'] ③
```

- ① Specifies that an external Red Hat Gluster Storage cluster is used.
- ② Specifies the IP address or host name of the external Heketi service.
- ③ The default storage plug-in dependencies to install. By default, the Ceph and Red Hat Gluster Storage plug-in dependencies are installed.

The following example shows a completed inventory file for externally hosted Red Hat Gluster Storage:

```
[OSEv3:children]
masters
nodes
glusterfs

[OSEv3:vars]
ansible_ssh_user=root
openshift_deployment_type=openshift-enterprise
openshift_storage_glusterfs_is_native=False
openshift_storage_glusterfs_heketi_url=172.0.0.1
osn_storage_plugin_deps=['ceph', 'glusterfs']

[glusterfs]
node0 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
node1 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
node2 glusterfs_devices='["/dev/vdb", "/dev/vdc", "/dev/vdd"]'
```

### Running the Playbook for External Red Hat Gluster Storage

To run the playbook for a natively hosted Red Hat Gluster Storage deployment:

```
[user@demo playbooks]$ ansible-playbook -i inventory-file playbook-file
```

## References

Further information is available in the chapter on Advanced Installation in the *Installation and Configuration Guide* for OpenShift Container Platform 3.6; at  
[https://access.redhat.com/documentation/en-us/openshift\\_container\\_platform/3.6](https://access.redhat.com/documentation/en-us/openshift_container_platform/3.6)

# Guided Exercise: Configuring Storage Providers

In this exercise, you will prepare to provision GlusterFS storage.

## Outcomes

You should be able to add plug-in, host, and device entries to an Ansible inventory file to manage the installation of Red Hat Gluster Storage.

## Before you begin

All chapter 4 guided exercises and labs must have been completed. If not, complete the guided exercises and labs in chapter four before running this exercise.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab storage-configure setup** command. The script ensures that the **console** VM is running and the required files are installed for this exercise:

```
[student@workstation ~]$ lab storage-configure setup
```

## Steps

1. Verify that the **hosts-ucf** inventory file is available on the **console** VM.

- 1.1. On the **workstation** VM, open a new terminal and use the **ssh** command to log in to **console** as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

- 1.2. Ensure that the **hosts-ucf** inventory file, created in the previous exercise, exists in the **/root/openshift-install/lab.example.com** directory on the **console** VM:

```
[root@console ~]# cd /root/openshift-install/lab.example.com
[root@console lab.example.com]# ls
hosts-ucf README.md rootCA.crt rootCA.csr rootCA.key
```

2. Add entries to the **hosts-ucf** inventory file that will be used with an Ansible Playbook to deploy Red Hat Gluster Storage on an existing cluster.

- 2.1. As the **root** user on the **console** VM, edit the **/root/openshift-install/lab.example.com/hosts-ucf** file.

Add **glusterfs** to the **[OSEv3:children]** section of the inventory file to enable the **[glusterfs]** group:

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
```

**glusterfs**

- 2.2. Define in the **[glusterfs]** group which nodes are going to host Red Hat Gluster Storage pods. At a minimum, each node must have a **glusterfs\_devices** variable defined. This variable defines a list of block devices that the node will have access to, and which are intended solely for use as storage.

**Note**

Each entry in the **glusterfs** section is a single line.

**Note**

If you prefer, you can copy the details from the **/root/D0380/labs/storage-configure/gluster.txt** file in the **console** VM.

```
[glusterfs]
infra1.lab.example.com glusterfs_ip=172.25.250.21 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra2.lab.example.com glusterfs_ip=172.25.250.22 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra3.lab.example.com glusterfs_ip=172.25.250.23 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
```

- 2.3. The updated **hosts-ucf** inventory file should read as follows:

```
hosts file for lab.example.com cluster
[OSEv3:children]
masters
etcd
nodes
nfs
lb
glusterfs

...output omitted...

[glusterfs]
infra1.lab.example.com glusterfs_ip=172.25.250.21 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra2.lab.example.com glusterfs_ip=172.25.250.22 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra3.lab.example.com glusterfs_ip=172.25.250.23 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
```

3. From **workstation**, open a new terminal and run the **lab storage-configure grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab storage-configure grade
...
Comparing Entries in [OSEv3:vars]
```

---

|                                           |      |
|-------------------------------------------|------|
| · Checking openshift_deployment_type..... | PASS |
| · Checking deployment_type.....           | PASS |
| · Checking openshift_release.....         | PASS |
| · Checking osm_cluster_network_cidr.....  | PASS |
| · Checking openshift_portal_net.....      | PASS |
| ...                                       |      |

4. Push the changes to the Git repository.

4.1. From the **console** VM, add and commit the updated inventory file to the Git repository:

```
[root@console lab.example.com]# cd /root/openshift-install
[root@console openshift-install]# git commit \
-a -m "Update for glusterfs configuration"
```

4.2. Push your changes to the remote repository:

```
[root@console openshift-install]# git push
```

5. This concludes this guided exercise.

# Configuring and Testing Red Hat Gluster Container-Native Storage

## Objective

After completing this section, students should be able to configure and test the Red Hat Gluster Storage Container-Native Storage (CNS) provider in an HA cluster.

## Configuring Storage with Red Hat Gluster Storage

OpenShift Container Platform can use Red Hat Gluster Storage CNS to support use cases that require flexible storage for a container and large amounts of data such as log files or an OpenShift Container Platform integrated registry. To deploy Red Hat Gluster Storage CNS use the Ansible Playbook from the advanced installation method. Although you can install Red Hat Gluster Storage CNS manually, Red Hat strongly discourages this approach due to the excessive amount of file configuration that are required.

### General Purpose Storage with Red Hat Gluster Storage

To configure general purpose storage with Red Hat Gluster Storage, add a **glusterfs** entry to the **[OSEv3:children]** group.

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
glusterfs
```

Create an additional group named **[glusterfs]**. Group entries must list the set of hosts and the devices that Red Hat Gluster Storage uses to store data:

```
[glusterfs]
infra1.lab.example.com glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
glusterfs_ip=172.25.250.21
infra2.lab.example.com glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
glusterfs_ip=172.25.250.22
infra3.lab.example.com glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
glusterfs_ip=172.25.250.23
```

This configuration automatically configures the Red Hat Gluster Storage daemon set and the Heketi API to manage the pods and the Red Hat Gluster Storage devices. Also, Red Hat Gluster Storage configures the storage class named **glusterfs-storage**, which may be referenced during persistent volume creation.

To install the Red Hat Gluster Storage CNS, run the following command:

```
[root@demo ~]# ansible-playbook -i inventory-file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-glusterfs/config.yml
```

### Integrated Registry with Red Hat Gluster Storage

To configure storage for the integrated registry to use Red Hat Gluster Storage, add a **glusterfs\_registry** entry to the **[OSEv3:children]** group.

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
glusterfs
glusterfs_registry
```

In the previous section, the Ansible inventory file was configured for the integrated registry to be hosted on an NFS share. To change this configuration to use Red Hat Gluster Storage, remove or comment the lines that configured the registry to use the NFS storage:

```
#openshift_hosted_registry_storage_kind=nfs
#openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
#openshift_hosted_registry_storage_nfs_directory=/exports
#openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
#openshift_hosted_registry_storage_volume_name=registry
#openshift_hosted_registry_storage_volume_size=10Gi
```

To use Red Hat Gluster Storage for the hosted registry, add the **openshift\_hosted\_registry\_storage\_kind** variable to the **[OSEv3:vars]** section, and set its value to **glusterfs**.

```
openshift_hosted_registry_storage_kind=glusterfs
```

Edit the Ansible inventory file to add the **[glusterfs\_registry]** group to the end of the file. Include the nodes in each group that will host the Red Hat Gluster Storage:

```
[glusterfs]
infra1.lab.example.com glusterfs_ip=172.25.250.21 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra2.lab.example.com glusterfs_ip=172.25.250.22 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra3.lab.example.com glusterfs_ip=172.25.250.23 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'

[glusterfs_registry]
infra1.lab.example.com glusterfs_ip=172.25.250.21 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra2.lab.example.com glusterfs_ip=172.25.250.22 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra3.lab.example.com glusterfs_ip=172.25.250.23 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
```

To install GlusterFS for the integrated registry, use the following command:

```
[root@demo ~]# ansible-playbook -i inventory-file \
/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml
```

The inventory file ensures the configuration of the Red Hat Gluster Storage daemon set, Heketi API, and the **gluster-storage** storage class. Additionally, it references inventory files for deploying general purpose and registry storage using Red Hat Gluster Storage.

To switch the integrated registry persistent volume claim, create a new volume claim **gluster-registry-claim**. To do so, the following YAML configuration may be created:

```
apiVersion: v1
kind: PersistentVolumeClaim
metadata:
 name: gluster-registry-claim
spec:
 accessModes:
 - ReadWriteMany
 resources:
 requests:
 storage: 10Gi
 storageClassName: glusterfs-storage
```

In order to swap the requests from the existing NFS to the new GlusterFS, update the **docker-registry** deployment configuration by executing the following command:

```
[root@demo ~]# oc volume dc/docker-registry \
--add --name=registry-storage -t pvc \
--claim-name=gluster-registry-claim --overwrite
```

## References

Red Hat Gluster Storage Ansible Playbook installer documentation.  
<https://github.com/openshift/openshift-ansible/tree/master/playbooks/openshift-glusterfs>

Further information is available in the *Complete Example Using GlusterFS* chapter of the *OpenShift Container Platform Installation and Configuration Guide* for OpenShift 3.6; at  
<https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Configuring and Testing Red Hat Gluster Storage CNS

In this exercise, you will configure an inventory file with entries for Red Hat Gluster Storage and run an Ansible playbook to install Container-Native Storage (CNS) with Red Hat Gluster for use in an OpenShift Container Platform environment.

## Outcomes

You should be able to:

- Add group and host entries to an inventory file for deploying CNS with Red Hat Gluster.
- Run an Ansible Playbook to deploy CNS with Red Hat Gluster Storage.
- Launch an application that uses Red Hat Gluster Storage.

## Before you begin

The guided exercise from the section *the section called “Guided Exercise: Configuring Storage Providers”* must have been completed. If not, complete the guided exercise before running this exercise.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab storage-cns setup** command. The script ensures connectivity with the **console** and **services** VMs and that the required files are installed for this exercise:

```
[student@workstation ~]$ lab storage-cns setup
```

## Steps

1. Verify that the Ansible inventory file is available on the **console** VM.

1. On the **workstation** VM, open a new terminal and use the **ssh** command to log in to **console** as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

2. Go to the **openshift-install** directory and run the **git pull** command to ensure that the files are current.

```
[root@console ~]# cd openshift-install
[root@console openshift-install]# git pull
Already up-to-date.
```

3. Ensure that the **hosts-ucf** Ansible inventory file, created in the previous exercise, exists in the **/root/openshift-install/lab.example.com/** directory on the **console** VM:

```
[root@console openshift-install]# cd lab.example.com
[root@console lab.example.com]# ls
```

```
rootCA.crt rootCA.csr rootCA.key hosts-ucf README.MD
```

2. Inspect the **hosts-ucf** inventory file and verify that the **glusterfs** entry is listed in the **[OSEv3:children]** group:

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
glusterfs
```

3. Inspect the **hosts-ucf** inventory file and verify that the **[glusterfs]** group exists and the entries for the three infrastructure nodes are listed:

```
[glusterfs]
infra1.lab.example.com glusterfs_ip=172.25.250.21 glusterfs_devices='["/dev/vdc",
"/dev/vdd", "/dev/vde"]'
infra2.lab.example.com glusterfs_ip=172.25.250.22 glusterfs_devices='["/dev/vdc",
"/dev/vdd", "/dev/vde"]'
infra3.lab.example.com glusterfs_ip=172.25.250.23 glusterfs_devices='["/dev/vdc",
"/dev/vdd", "/dev/vde"]'
```

4. Run the Ansible Playbook to configure Red Hat Gluster Storage.

- 4.1. Run the **/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-glusterfs/config.yml** Ansible Playbook with the **hosts-ucf** inventory file:

```
[root@console lab.example.com]# ansible-playbook -i hosts-ucf \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-glusterfs/ \
config.yml
```



## Note

The following output is expected during the Ansible Playbook execution:

```
TASK [openshift_storage_glusterfs : Wait for GlusterFS pods] ****
FAILED - RETRYING: Wait for GlusterFS pods (30 retries left).
FAILED - RETRYING: Wait for GlusterFS pods (29 retries left).
FAILED - RETRYING: Wait for GlusterFS pods (28 retries left).
FAILED - RETRYING: Wait for GlusterFS pods (27 retries left).
```

5. To test the Red Hat Gluster Storage installation, deploy an application from the **workstation** VM.

To test for successful deployment of Red Hat Cluster Storage, deploy an Nginx application.

- 5.1. From the **workstation** VM, log in to the OpenShift cluster as the **admin** user with **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

5.2. Navigate to the **/home/student/D0380/labs/storage-cns** directory:

```
[student@workstation ~]$ cd /home/student/D0380/labs/storage-cns
[student@workstation storage-cns]$
```

5.3. List the contents of the **/home/student/D0380/labs/storage-cns** directory.

The directory contains the Nginx image stream and the template required for the application:

```
[student@workstation storage-cns]$ ls
nginx-gluster-template.yml nginx-slim-is.yml
```

5.4. Deploy the **nginx-gluster** template to the OpenShift cluster:

```
[student@workstation storage-cns]$ oc create -f nginx-gluster-template.yml \
-n openshift
template "nginx-gluster" created
```

5.5. Create the **nginx-slim** image stream to deploy the container image:

```
[student@workstation storage-cns]$ oc create -f nginx-slim-is.yml \
-n openshift
imagestream "nginx-slim" created
```

5.6. Import the **nginx-slim** image to deploy the application:

```
[student@workstation storage-cns]$ oc import-image nginx-slim:0.8 \
-n openshift
The import completed successfully.
...output omitted...
```

5.7. Create a temporary project for the Nginx application.

On the **workstation** VM, run the **oc new-project** command:

```
[student@workstation storage-cns]$ oc new-project teststorage
```

5.8. Relax the security in the cluster. This allows images to run without a preallocated UID. Grant all authenticated users access to the **anyuid** security context constraint (SCC):

```
[student@workstation storage-cns]$ oc adm policy add-scc-to-user \
anyuid -z default
```

5.9. Launch the **nginx-slim** application:

```
[student@workstation storage-cns]$ oc new-app --template nginx-gluster \
--param=WEBSERVER_SERVICE_NAME=nginx1
--> Deploying template "openshift/nginx-gluster" to project teststorage

 Nginx (Persistent)

 The Google slim nginx container with the html directory persisted to
 GlusterFS.
 NOTE: You must have GlusterFS CNS configured to use this template.

 An nginx server has been created with persistent storage for the html files
 using a dynamically allocated persistent volume.

 * With parameters:
 * Webserver Service Name=nginx1
 * Volume Capacity=1Gi

--> Creating resources ...
service "nginx1" created
persistentvolumeclaim "nginx1-claim" created
deploymentconfig "nginx1" created
--> Success
Run 'oc status' to view your app.
```

6. Run the following commands to verify that RedHat Gluster Storage is used by the **nginx-slim** application.

- 6.1. Run the following command to retrieve the storage class:

```
[student@workstation storage-cns]$ oc get storageclass
NAME TYPE
glusterfs-storage kubernetes.io/glusterfs
```

- 6.2. To verify the status of the **nginx-slim** application, run the following command:

```
[student@workstation storage-cns]$ oc status
In project teststorage on server https://console.lab.example.com:443

svc/glusterfs-dynamic-nginx1-claim - 172.30.177.120:1

svc/nginx1 - 172.30.54.157:80
dc/nginx1 deploys openshift/nginx-slim:0.8
 deployment #1 deployed 4 hours ago - 1 pod

View details with 'oc describe <resource>/<name>' or list everything with 'oc
get all'.
```

- 6.3. Retrieve the list of the persistent volume claims in the **teststorage** project to ensure that a volume has been provisioned in the Gluster cluster. The **STATUS** should read **Bound**.

```
[student@workstation storage-cns]$ oc get pvc -n teststorage
NAME STATUS VOLUME
nginx1-claim Bound pvc-0aaac536-d92b-11e7-aa81-52540000fa0d
CAPACITY ACCESSMODES STORAGECLASS AGE
1Gi RWX glusterfs-storage 2m
```

---

7. Clean up by removing the **teststorage** project.

7.1. Delete the **teststorage** project:

```
[student@workstation storage-configure-test]$ oc delete project teststorage
```

This concludes the guided exercise.

# Lab: Provisioning Persistent Storage

In this lab, you will modify an existing Ansible inventory to install Red Hat Gluster Storage that is used by the local registry.

## Outcomes

You should be able to:

- Verify entries in an existing inventory file to configure Red Hat Gluster Storage for the local OpenShift registry.
- Run an Ansible Playbook to deploy Red Hat Gluster Storage for the local OpenShift registry.
- Launch an application to verify that the local registry uses Red Hat Gluster Storage.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab storage-review setup** command. The script ensures that the cluster hosts are reachable and that the required files are installed for this exercise:

```
[student@workstation ~]$ lab storage-review setup
```

## Steps

- As the **root** user on the **console** VM, clone or pull the **openshift-install** Git repository from the **services.lab.example.com** VM.
- Modify the **hosts-ucf** Ansible inventory file to add the **glusterfs** and **glusterfs\_registry** entries to the **[OSEv3:children]** section to enable the **[glusterfs]** and **[glusterfs\_registry]** groups.

Comment the lines that configure the registry to use NFS storage and configure the registry to use Red Hat Gluster CNS. Add the variable **openshift\_storage\_glusterfs\_wipe=True** to the **[OSEv3:vars]** group.

Create the **[glusterfs\_registry]** group and define the entries according to the following table:

| Variable                      | Value                                                                                                  |
|-------------------------------|--------------------------------------------------------------------------------------------------------|
| <b>infra1.lab.example.com</b> | <b>glusterfs_ip=172.25.250.21</b><br><b>glusterfs_devices='[ "/dev/vdc", "/dev/vdd", "/dev/vde" ]'</b> |
| <b>infra2.lab.example.com</b> | <b>glusterfs_ip=172.25.250.22</b><br><b>glusterfs_devices='[ "/dev/vdc", "/dev/vdd", "/dev/vde" ]'</b> |
| <b>infra3.lab.example.com</b> | <b>glusterfs_ip=172.25.250.23</b><br><b>glusterfs_devices='[ "/dev/vdc", "/dev/vdd", "/dev/vde" ]'</b> |

- 
3. From **workstation**, open a new terminal and run the **lab storage-review check** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.
  4. From the **console** VM, add the inventory file to the index and commit your changes before pushing them to the Git repository. Use a message of **Integrates Gluster CNS for the registry**.
  5. Run the **/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml** Ansible Playbook with the **hosts-ucf** inventory file to update the environment with Red Hat Gluster Storage for the local registry additions.



### Note

The installation takes about 25 to 30 minutes to complete.



### Note

The installation can fail with the following message:

```
{u'returncode': 1, u'cmd': u'/usr/bin/oc replace -f \ /tmp/openshift-glusterfs-ansible-oAsQwF/glusterfs-registry-service.yml -n default', \ u'results': {}, u'standard_error': \ u'The Service "glusterfs-registry-endpoints" is invalid: spec.clusterIP: \ Invalid value: "": field is immutable\n', u'standard_out': u''}
```

To resolve this issue, run the following command as the OpenShift cluster **admin** user.

```
[student@workstation ~]$ oc delete svc glusterfs-registry-endpoints -n default
```

6. From the **workstation** VM, configure the hosted registry to use the Red Hat Gluster Storage CNS as the back end.
  - 6.1. Log in as the **admin** user with the **redhat** password to the OpenShift cluster.
  - 6.2. Change to the **default** project.
  - 6.3. Create the volume claim **gluster-registry-claim** by using the YAML file located at **/home/student/D0380/labs/storage-review/gluster-registry-claim.yaml**.
  - 6.4. Retrieve the list of persistent volume claims to ensure that the claim is bound to a persistent volume.
  - 6.5. Update the **docker-registry** deployment configuration by swapping the existing registry storage volume for the new Red Hat Gluster Storage volume by executing the following command:

```
[student@workstation ~]$ oc volume dc/docker-registry \
--add --name=registry-storage -t pvc \
--claim-name=gluster-registry-claim --overwrite
deploymentconfig "docker-registry" updated
```

The update triggers the deployment of new pods.

7. Verify the changes applied to the deployment configuration.
  - 7.1. On the **workstation** VM, as the **student** user, retrieve the list of the pods running in the **glusterfs** project.
  - 7.2. Replace the pod name in the following command with one from the list in the previous command's output to connect to a **glusterfs** storage pod.
  - 7.3. From the Red Hat Gluster Storage server, run the **gluster volume list** command to ensure that a new volume named **vol\_d7d1e83234aad471ac2b034b8d1c3cd4** is present for the registry and exit the server.
  - 7.4. Retrieve the list of the registry pods for the in the **default** project.  
If there are only **docker-registry-1** pods running, and the docker-registry-2-\* is in **ContainerCreating** status, review any of the previous steps for typos.
  - 7.5. Use the **rsh** of the **oc** command to log in to one of the registry pods listed in the previous step.
  - 7.6. Ensure that the **registry** volume mounted on the system's root directory is connected to Red Hat Gluster Storage by running the **mount** command. Notice the UUID of the volume name which should match the volume name in the storage cluster. Exit the server.
8. As the **student** user on the **workstation** VM, run **lab storage-review grade** to grade your work:

```
[student@workstation ~]$ lab storage-review grade
```

9. Clean up the lab:

```
[student@workstation ~]$ lab storage-review cleanup
```

This concludes the lab.

# Solution

In this lab, you will modify an existing Ansible inventory to install Red Hat Gluster Storage that is used by the local registry.

## Outcomes

You should be able to:

- Verify entries in an existing inventory file to configure Red Hat Gluster Storage for the local OpenShift registry.
- Run an Ansible Playbook to deploy Red Hat Gluster Storage for the local OpenShift registry.
- Launch an application to verify that the local registry uses Red Hat Gluster Storage.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab storage-review setup** command. The script ensures that the cluster hosts are reachable and that the required files are installed for this exercise:

```
[student@workstation ~]$ lab storage-review setup
```

## Steps

- As the **root** user on the **console** VM, clone or pull the **openshift-install** Git repository from the **services.lab.example.com** VM.
  - On the **workstation** VM, open a new terminal and use the **ssh** command to log in to **console** VM as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

- Pull the **openshift-install** Git repository from **services.lab.example.com** VM if you have not done that in the previous guided exercises:

```
[root@console ~]# cd openshift-install
[root@console openshift-install]# git pull
```

- Modify the **hosts-ucf** Ansible inventory file to add the **glusterfs** and **glusterfs\_registry** entries to the **[OSEv3:children]** section to enable the **[glusterfs]** and **[glusterfs\_registry]** groups.

Comment the lines that configure the registry to use NFS storage and configure the registry to use Red Hat Gluster CNS. Add the variable **openshift\_storage\_glusterfs\_wipe=True** to the **[OSEv3:vars]** group.

Create the **[glusterfs\_registry]** group and define the entries according to the following table:

| Variable                            | Value                                                                                                  |
|-------------------------------------|--------------------------------------------------------------------------------------------------------|
| <code>infra1.lab.example.com</code> | <code>glusterfs_ip=172.25.250.21<br/>glusterfs_devices='[ "/dev/vdc", "/dev/vdd", "/dev/vde" ]'</code> |
| <code>infra2.lab.example.com</code> | <code>glusterfs_ip=172.25.250.22<br/>glusterfs_devices='[ "/dev/vdc", "/dev/vdd", "/dev/vde" ]'</code> |
| <code>infra3.lab.example.com</code> | <code>glusterfs_ip=172.25.250.23<br/>glusterfs_devices='[ "/dev/vdc", "/dev/vdd", "/dev/vde" ]'</code> |

- 2.1. Navigate to the `/root/openshift-install/lab.example.com` directory and list the directory contents:

```
[root@console openshift-install]# cd lab.example.com
[root@console lab.example.com]# ls
hosts-ucf README.MD rootCA.crt rootCA.csr rootCA.key
```

- 2.2. Edit the `hosts-ucf` inventory file to add the `glusterfs` and `glusterfs_registry` entries to the `[OSEv3:children]` section to enable the `[glusterfs]` and `[glusterfs_registry]` groups:

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
glusterfs
glusterfs_registry
```

- 2.3. Comment the lines that configure the registry to use NFS storage.

The commented lines should match the following entries:

```
openshift_hosted_registry_storage_kind=nfs
openshift_hosted_registry_storage_access_modes=['ReadWriteMany']
openshift_hosted_registry_storage_nfs_directory=/exports
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_registry_storage_volume_name=registry
openshift_hosted_registry_storage_volume_size=10Gi
```

- 2.4. Add the `openshift_hosted_registry_storage_kind` variable to the `[OSEv3:vars]` section and set its value to `glusterfs`.

```
openshift_hosted_registry_storage_kind=glusterfs
```

- 2.5. Add the `openshift_storage_glusterfs_wipe` variable to the `[OSEv3:vars]` section and set its value to `True`.

```
openshift_storage_glusterfs_wipe=True
```

- 2.6. Edit the **hosts-ucf** inventory file to add the **[glusterfs\_registry]** group to the end of the file. Include the nodes in the group that will host RedHat Gluster Storage:



### Note

Each entry in the group is a single line.

```
[glusterfs_registry]
infra1.lab.example.com glusterfs_ip=172.25.250.21 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra2.lab.example.com glusterfs_ip=172.25.250.22 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
infra3.lab.example.com glusterfs_ip=172.25.250.23 glusterfs_devices='["/dev/vdc", "/dev/vdd", "/dev/vde"]'
```



### Note

The file is available as **/root/D0380/labs/storage-review/glusterfs.txt** on the **console** VM.

3. From **workstation**, open a new terminal and run the **lab storage-review check** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab storage-review check
...
Comparing Entries in [OSEv3:vars]
 · Checking openshift_deployment_type..... PASS
 · Checking deployment_type..... PASS
 · Checking openshift_release..... PASS
 · Checking osm_cluster_network_cidr..... PASS
 · Checking openshift_portal_net..... PASS
...
```

4. From the **console** VM, add the inventory file to the index and commit your changes before pushing them to the Git repository. Use a message of **Integrates Gluster CNS for the registry**.

```
[root@console lab.example.com]# git add .
[root@console lab.example.com]# git commit -m \
"Integrates Gluster CNS for the registry"
[root@console lab.example.com]# git push
...
Counting objects: 7, done.
Delta compression using up to 2 threads.
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 679 bytes | 0 bytes/s, done.
```

```
Total 4 (delta 1), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install
 7b9958c..2ec0d1e master -> master
```

5. Run the **/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml** Ansible Playbook with the **hosts-ucf** inventory file to update the environment with Red Hat Gluster Storage for the local registry additions.



### Note

The installation takes about 25 to 30 minutes to complete.



### Note

The installation can fail with the following message:

```
{u'returncode': 1, u'cmd': u'/usr/bin/oc replace -f \
/tmp/openshift-glusterfs-ansible-oAsQwF/glusterfs-registry-service.yml -n
default', \
u'results': {}, u'stderr': \
u'The Service "glusterfs-registry-endpoints" is invalid: spec.clusterIP: \
Invalid value: "": field is immutable\n', u'stdout': u''}
```

To resolve this issue, run the following command as the OpenShift cluster **admin** user.

```
[student@workstation ~]$ oc delete svc glusterfs-registry-endpoints -n
default
```

- 5.1. Use the following command to run the Ansible Playbook:

```
[root@console lab.example.com]# ansible-playbook -i hosts-ucf \
/usr/share/ansible/openshift-ansible/playbooks/byo/config.yml
...output omitted...
PLAY RECAP ****
apps1.lab.example.com : ok=251 changed=59 unreachable=0 failed=0
apps2.lab.example.com : ok=251 changed=59 unreachable=0 failed=0
console.lab.example.com : ok=128 changed=22 unreachable=0 failed=0
infra1.lab.example.com : ok=283 changed=61 unreachable=0 failed=0
infra2.lab.example.com : ok=283 changed=61 unreachable=0 failed=0
infra3.lab.example.com : ok=283 changed=61 unreachable=0 failed=0
localhost : ok=15 changed=0 unreachable=0 failed=0
master1.lab.example.com : ok=806 changed=230 unreachable=0 failed=0
master2.lab.example.com : ok=468 changed=126 unreachable=0 failed=0
master3.lab.example.com : ok=468 changed=125 unreachable=0 failed=0
```

- 5.2. Exit from the **console** VM:

```
[root@console lab.example.com]# exit
[student@workstation ~]$
```

6. From the **workstation** VM, configure the hosted registry to use the Red Hat Gluster Storage CNS as the back end.

- 6.1. Log in as the **admin** user with the **redhat** password to the OpenShift cluster.

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

 default
 glusterfs
 kube-public
 kube-system
 logging
 management-infra
 openshift
 openshift-infra
* teststorage

Using project "teststorage".
```

- 6.2. Change to the **default** project.

Run the following command to change to the **default** project:

```
[student@workstation ~]$ oc project default
Now using project "default" on server "https://console.lab.example.com:443".
```

- 6.3. Create the volume claim **gluster-registry-claim** by using the YAML file located at **/home/student/D0380/labs/storage-review/gluster-registry-claim.yaml**.

```
[student@workstation ~]$ oc create -f \
/home/student/D0380/labs/storage-review/gluster-registry-claim.yaml
persistentvolumeclaim "gluster-registry-claim" created
```

- 6.4. Retrieve the list of persistent volume claims to ensure that the claim is bound to a persistent volume.

```
[student@workstation ~]$ oc get pvc
NAME STATUS VOLUME
...output omitted...
gluster-registry-claim Bound pvc-509ca056-d942-11e7-aa81-52540000fa0d
 CAPACITY ACCESSMODES STORAGECLASS AGE
 10Gi RWX glusterfs-storage 31m
```

- 6.5. Update the **docker-registry** deployment configuration by swapping the existing registry storage volume for the new Red Hat Gluster Storage volume by executing the following command:

```
[student@workstation ~]$ oc volume dc/docker-registry \
--add --name=registry-storage -t pvc \
```

```
--claim-name=gluster-registry-claim --overwrite
deploymentconfig "docker-registry" updated
```

The update triggers the deployment of new pods.

7. Verify the changes applied to the deployment configuration.

- 7.1. On the **workstation** VM, as the **student** user, retrieve the list of the pods running in the **glusterfs** project.

```
[student@workstation ~]$ oc get pods -n glusterfs
NAME READY STATUS RESTARTS AGE
glusterfs-storage-d6j4g 1/1 Running 0 19h
glusterfs-storage-fzwmwh 1/1 Running 0 19h
glusterfs-storage-wbrzx 1/1 Running 0 19h
heketi-storage-1-dfnx1 1/1 Running 0 19h
```

- 7.2. Replace the pod name in the following command with one from the list in the previous command's output to connect to a **glusterfs** storage pod.

```
[student@workstation ~]$ oc rsh -n glusterfs glusterfs-storage-d6j4g
sh-4.2#
```

- 7.3. From the Red Hat Gluster Storage server, run the **gluster volume list** command to ensure that a new volume named **vol\_d7d1e83234aad471ac2b034b8d1c3cd4** is present for the registry and exit the server.

```
sh-4.2# gluster volume list
...
vol_d7d1e83234aad471ac2b034b8d1c3cd4
sh-4.2# exit
```

- 7.4. Retrieve the list of the registry pods for the in the **default** project.

```
[student@workstation ~]$ oc get pods -n default
...
NAME READY STATUS RESTARTS AGE
docker-registry-2-3sw4d 1/1 Running 0 34m
docker-registry-2-9xqn5 1/1 Running 0 34m
docker-registry-2-lgsck 1/1 Running 0 34m
...
```

If the there are only **docker-registry-1** pods running, and the docker-registry-2-\* is in **ContainerCreating** status, review any of the previous steps for typos.

- 7.5. Use the **rsh** of the **oc** command to log in to one of the registry pods listed in the previous step.

```
[student@workstation ~]$ oc rsh docker-registry-2-3sw4d
```

- 7.6. Ensure that the **registry** volume mounted on the system's root directory is connected to Red Hat Gluster Storage by running the **mount** command. Notice the UUID of the

volume name which should match the volume name in the storage cluster. Exit the server.

```
sh-4.2# mount | grep registry
...
172.25.250.21:vol_d7d1e83234aad471ac2b034b8d1c3cd4 \
on /registry type fuse.glusterfs \
(rw,relatime,user_id=0,group_id=0, \
default_permissions,allow_other,max_read=131072)
...
sh-4.2# exit
```

8. As the **student** user on the **workstation** VM, run **lab storage-review grade** to grade your work:

```
[student@workstation ~]$ lab storage-review grade
```

9. Clean up the lab:

```
[student@workstation ~]$ lab storage-review cleanup
```

This concludes the lab.

## Summary

In this chapter, you learned:

- OpenShift Container Platform uses the Kubernetes persistent volume (PV) framework to allow administrators to provision persistent storage for a cluster. OpenShift cluster users can define persistent volume claims (PVCs) to request PV resources without having specific knowledge of the underlying storage infrastructure.
- The **StorageClass** resource object describes and classifies storage that can be requested by PVCs and provides a means for passing parameters for dynamically provisioned storage on demand. **StorageClass** objects also serve as a management mechanism for controlling different levels of storage and access to the storage.
- Dynamic provisioning enables provisioning of a Red Hat Gluster Storage volume to a running application container without having to create the volume beforehand.
- Red Hat Gluster Storage is a software-only, scale-out storage solution that provides flexible and agile unstructured data storage.
- Container-Native Storage (CNS) is a solution where applications require both shared file storage and the flexibility of a converged infrastructure with compute and storage instances that are scheduled and run from the same set of hardware.
- Using Ansible Playbooks and the advanced installation method, you can deploy CNS for either natively hosted containerized GlusterFS storage, or externally hosted Red Hat Gluster Storage.



## CHAPTER 6

# ENABLING LOG AGGREGATION

| Overview          |                                                                                                                                                                                            |
|-------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Goal</b>       | Enable the log aggregation feature.                                                                                                                                                        |
| <b>Objectives</b> | <ul style="list-style-type: none"><li>Describe the log aggregation feature.</li><li>Install the log aggregation feature in an HA cluster using the advanced installation method.</li></ul> |
| <b>Sections</b>   | <ul style="list-style-type: none"><li>Describing Log Aggregation (and Quiz)</li><li>Enabling Log Aggregation in an HA Cluster (and Guided Exercise)</li></ul>                              |
| <b>Lab</b>        | Enabling Log Aggregation                                                                                                                                                                   |

# Describing Log Aggregation

## Objective

After completing this section, students should be able to describe the log aggregation feature.

## Introduction to Log Aggregation

Managing the life cycle of an OpenShift cluster involves many tasks. One of these tasks is log management, which includes a log rotation policy and log aggregation. Each component in the cluster has its own set of logs, which can make log management a complicated and tedious task. For instance, you are likely to spend a lot of time tracing the root cause of a router that crashes deployed on multiple nodes if no solution is deployed for log management.

You can use a centralized system to facilitate troubleshooting. Many centralized log aggregators exist. OpenShift Container Platform provides a solution called *EFK*. The EFK installation is provided as an OpenShift Ansible Playbook and it is deployed as pods on OpenShift Container Platform.

EFK, is an initialism composed of the letter of three open source projects: *Elasticsearch*, *Fluentd*, and *Kibana*:

- Elasticsearch, an open source search engine and object store that provides a distributed RESTful API for logs
- Fluentd, a data collector project that gathers logs from the application nodes and sends them to the Elasticsearch service
- Kibana, a web interface for Elasticsearch

You can benefit from features such as user management for log access, graphs and charts, a quick overview of common errors, searching, and filtering by deploying an EFK stack in your environment. The EFK stack aggregates logs from all nodes and projects in the Elasticsearch database, and provides a web interface via Kibana to access the logs.

### Requirements for EFK Deployment

Ensure that the following prerequisites are met before running the Ansible Playbook that installs and configures the EFK stack.

- The cluster must satisfy the sizing recommendations, as referred to in the references section.
- A router must be present in the environment.
- A storage back end must be deployed for Elasticsearch, which requires its own storage volume.
- An OpenShift project must be created for the EFK stack.



## Note

The stack collects logs across all projects within OpenShift Container Platform. If the project does not exist, the Ansible Playbook will create one. You only need to create a project if you need to add a node selector, otherwise, the **openshift-logging** role creates a new project.

### EFK Stack Variables

To override the default values for the EFK stack installation, configure the variables in the Ansible inventory file. The following table lists some of the available options. Consult the Aggregating Container Logs section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* for a detailed list of Ansible variables.

| <b>Ansible File Logging Variables</b>         |                                                                                                                                                                                                     |
|-----------------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Variable</b>                               | <b>Purpose</b>                                                                                                                                                                                      |
| <b>openshift_logging_use_ops</b>              | If set to <b>true</b> , configures a second Elasticsearch cluster and Kibana instance for operations logs. Fluentd splits logs between the main cluster and a cluster reserved for operations logs. |
| <b>openshift_logging_master_url</b>           | The URL for the Kubernetes master. Does not need to be public facing, but it must be accessible from within the cluster.                                                                            |
| <b>openshift_logging_namespace</b>            | The namespace into which aggregated logging services are deployed.                                                                                                                                  |
| <b>openshift_logging_curator_default_days</b> | The default minimum age, in days, that Curator uses for deleting log records. Curator is installed with the EFK stack, but it can also be installed as a separate container.                        |
| <b>openshift_logging_kibana_hostname</b>      | The external host name for web clients to reach the Kibana service.                                                                                                                                 |
| <b>openshift_logging_kibana_memory_limit</b>  | The amount of memory to allocate to Kibana, for example, 1Gi                                                                                                                                        |
| <b>openshift_logging_kibana_nodeselector</b>  | A node selector that specifies which nodes are eligible targets for deploying Kibana instances.                                                                                                     |
| <b>openshift_logging_fluentd_nodeselector</b> | A node selector that specifies which nodes are eligible targets for deploying Fluentd instances. Any node where Fluentd should run must have this label for the service to run and collect logs.    |
| <b>openshift_logging_es_cluster_size</b>      | Defines the number of Elasticsearch replicas to deploy in the cluster. Three                                                                                                                        |

| Ansible File Logging Variables                |                                                                                                                                                                                                              |
|-----------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                                               | or more replicas are recommended to ensure redundancy.                                                                                                                                                       |
| <b>openshift_logging_es_pvc_size</b>          | Defines the size of the persistent volume claim in <b>GiB</b> to create per Elasticsearch instance. If the value is omitted, no PVCs are created, and Elasticsearch instances use ephemeral storage instead. |
| <b>openshift_logging_curator_nodeselector</b> | The node selector specifies the nodes that are eligible targets for deploying Curator instances.                                                                                                             |



## Important

NFS storage is neither recommended nor supported for Elasticsearch.

If NFS storage is a requirement in the organization, you can allocate a large file on a volume to serve as a storage device and mount it locally on one host.

### Scaling Elasticsearch

Elasticsearch instances do not scale easily, due to the nature of persistent volumes and the way that Elasticsearch is configured to store its data and recover the cluster.

To scale up the number of Elasticsearch instances, modify the Ansible inventory file and rerun the logging playbook that is discussed in the next section (**openshift-logging.yml**). If the logging environment has custom changes, create a new deployment configuration to the cluster instead of reinstalling the stack.

### Fluentd

When deploying the logging environment, Fluentd is deployed as a *DaemonSet*. A **DaemonSet** is an OpenShift object which ensures that all nodes run a copy of a pod. By default, the Fluentd service reads log entries from the **/var/log/messages** and **/var/log/containers/container.log** files. However, you can also use the **systemd** journal as the log source.

You can update the following variables to configure Fluentd:

- **openshift\_logging\_use\_journal**: Configures Fluentd to list the driver that Docker uses. Defaults to empty.

For example, if Docker runs with the **--log-driver=journald**, Fluentd reads from the **systemd** journal. Otherwise, Fluentd assumes that Docker is using the **json-file** log driver, which reads from the **/var/log/** directory.

- **openshift\_logging\_journal\_read\_from\_head**: If set to **false**, Fluentd starts reading from the end of the journal, ignoring historical logs. Otherwise, Fluentd starts reading logs from the beginning of the journal.



## Important

Aggregated logging is only supported using the **journald** driver in Docker. Consult the Aggregating Container Logs section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* for more information.

You can configure Fluentd to send a copy of each log message to both the Elasticsearch instance running in OpenShift Container Platform, and to an external Elasticsearch instance. For example, if the organization requires two Elasticsearch instances, one for auditing purposes and one for development, Fluentd can send a copy of the log message to each Elasticsearch instance.



## Note

You can also configure Fluentd to send logs to an external log aggregator instead of the default Elasticsearch service. Consult the Aggregating Container Logs section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* for more information.

### Kibana

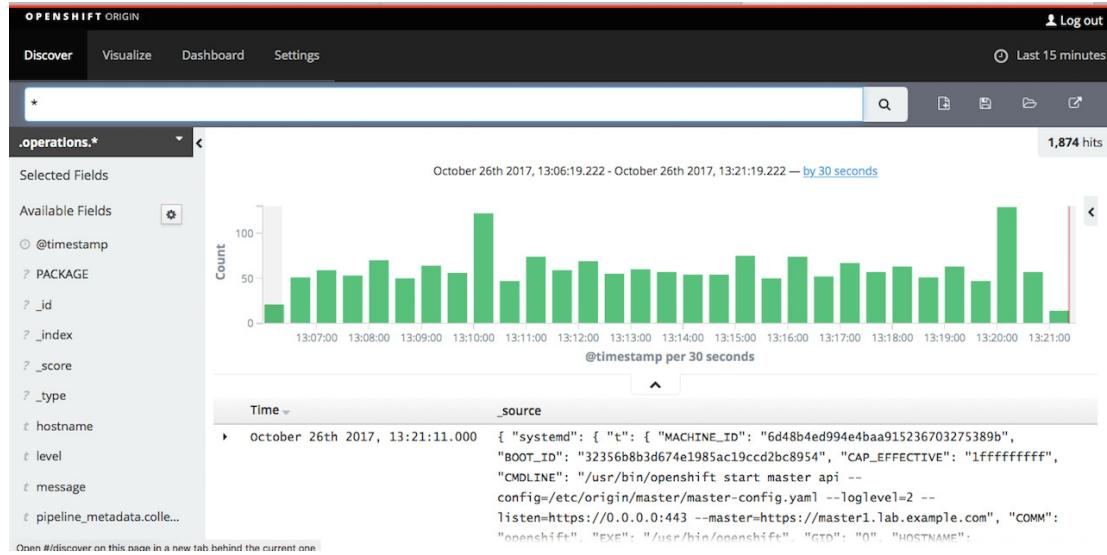
Kibana is the web interface that reads logs entries from the Elasticsearch database. It can create visualization graphs, charts, time tables, and reports, using time-based and non-time-based events. You can visualize the cluster data, export CSV files, create dashboards, and run advanced requests.

To access the Kibana web console after you deploy the EFK stack, add the **loggingPublicURL** variable to the `/etc/origin/master/master-config.yaml` configuration file. The value is the URL of the Kibana console, for example:

```
...
assetConfig:
...
loggingPublicURL: "https://kibana.example.com"
...
```

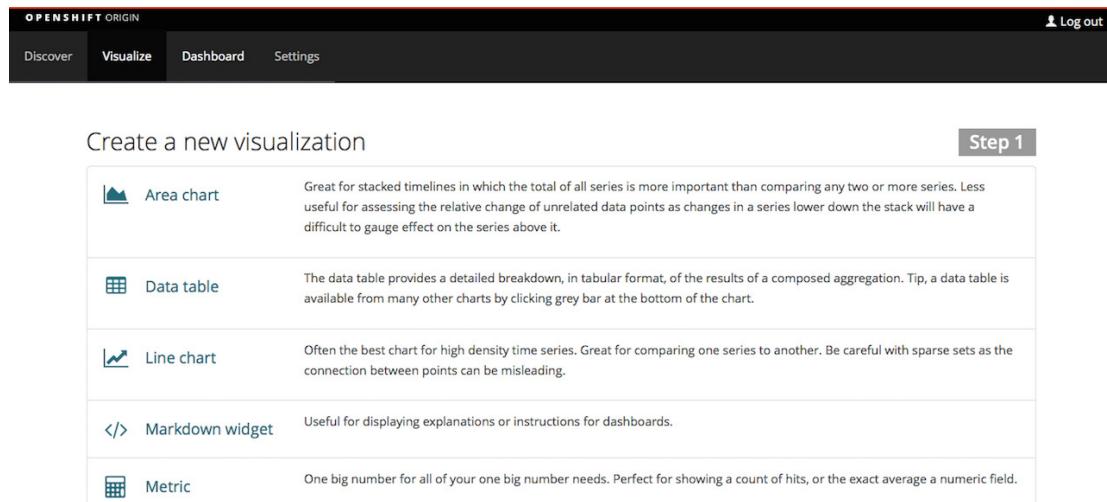
The *Figure 6.1: The Kibana web interface* shows some Kibana charts, available from the **Discover** menu, which describes the occurrences of log entries. The view has a search field that you can use to run searches with advanced patterns. For example, a search of **response:200** lists all documents that contain a **response:200** entry. Use an exclamation mark to negate a query.

You can use the **Discover** page to exclude values from a list, add columns to tables, or use the search bar to search for documents.



*Figure 6.1: The Kibana web interface*

Use the **Visualize** tab to generate visualizations, such as area charts, data tables, line charts, tile maps, and pie charts. You can use the search bar to update any visualization.



*Figure 6.2: The Visualize page*

You can save each graph and add it later on the dashboard. You can share entire dashboards as embedded iFrames or via a generated HTML link.

You can use the dashboard to update charts in real time, by using the date picker and the search bar. For each graph, tools such as filters, are available. You can also share a snapshot of the data retrieved in a current point in time by Kibana.

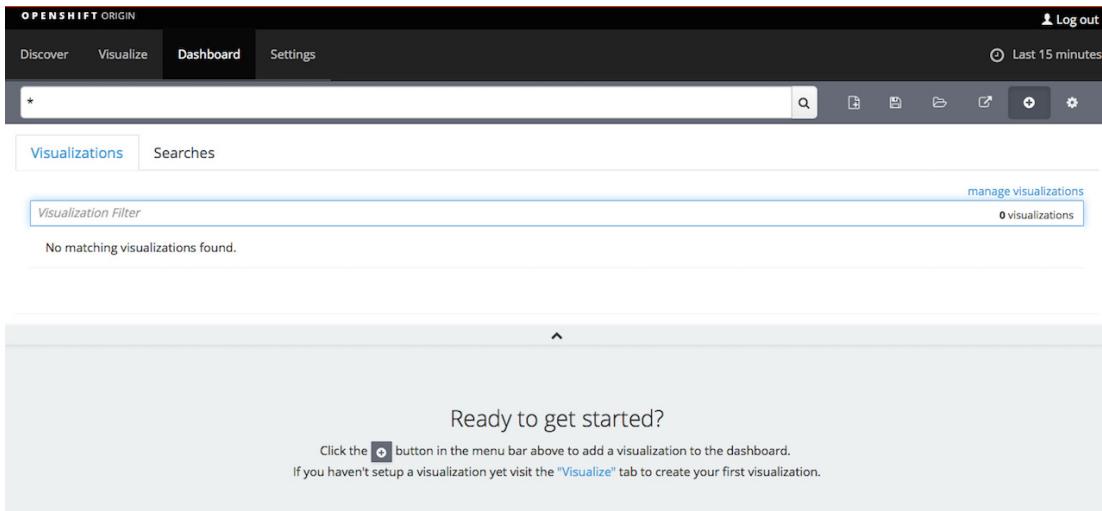


Figure 6.3: The Kibana dashboard

### Configuring Curator

Curator is the service that removes old indexes from Elasticsearch. You can use Curator to configure maintenance schedules for Elasticsearch on a per-project basis. Red Hat recommends that you configure only one Curator pod for each Elasticsearch cluster. The pod reads its configuration from a YAML file that is structured as follows:

```
PROJECT_NAME: ①
 ACTION: ②
 UNIT: VALUE ③
PROJECT_NAME:
 ACTION:
 UNIT: VALUE
 ...

```

- ① Defines the name of the project, such as **logging-devel**.
- ② Defines the action to take. Currently, **delete** is the only action allowed.
- ③ Defines the unit, such as days, weeks, or months.

Consider the following Curator requirements:

- Delete indexes in the **logging-devel** project that are older than one day.
- Delete **operations** logs that are older than eight weeks.
- Delete all other project indexes after 30 days.
- Run the jobs at midnight every day.

To configure Curator to address these requirements, create the following configuration file:

```
logging-devel:
 delete:
 days: 1

.operations:
 delete:
```

```
weeks: 8

.defaults:
 delete:
 days: 30
 runhour: 0
 runminute: 0
```

You can use the **openshift\_logging** Ansible role, which provides a configuration map, from which Curator reads its configuration. Edit the configuration map file to edit or replace the Curator configuration.

The following steps demonstrate how to update a Curator configuration:

1. Run the **oc edit** command to edit the configuration map object:

```
[root@demo ~]# oc edit configmap/logging-curatore
```

2. Run the **oc create** to replace the configuration map instead of editing it:

```
[root@demo ~]# oc create configmap logging-curatore -o yaml \
--from-file=config.yaml=/path/to/curatorconfig.yaml | \
oc replace -f
```

3. Run the **oc rollout** command to redeploy the Curator service:

```
[root@demo ~]# oc rollout latest dc/logging-curatore
[root@demo ~]# oc rollout latest dc/logging-curatore-ops
```

## References

Elasticsearch  
<http://elastic.co>

Kibana  
<https://www.elastic.co/products/kibana>

Fluentd  
<https://www.fluentd.org>

Further information is available in the Aggregate Logging Sizing Guidelines section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at  
<https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Aggregating Container Logs section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at  
<https://access.redhat.com/documentation/en-US/index.html>

# Quiz: Describing Log Aggregation

Choose the correct answers to the following questions:

1. Which of the following services is the data collector that gather logs from the application nodes?
  - a. Curator
  - b. Fluentd
  - c. Kibana
  - d. Elasticsearch
2. Which two storage back ends are recommended for use with Elasticsearch? (Choose two.)
  - a. GlusterFS
  - b. CephFS
  - c. NFS
  - d. CIFS
3. Which of the following objects is used for deploying Fluentd?
  - a. **LogConfig**
  - b. **ConfigMaps**
  - c. **DeploymentConfig**
  - d. **DaemonSet**
4. Which of the following lists describes the Curator configuration file presented below?

```
logging-devel:
 delete:
 days: 10

.operations:
 delete:
 days: 30

.defaults:
 delete:
 days: 30
 runhour: 0
 runminute: 0
```

- a.
  - Delete indexes in the **default** project older than 30 days.
  - Delete all other project indexes after 10 days.
  - Run the Curator jobs at midnight every day.
- b.
  - Delete indexes in the **logging-devel** project older than 10 days.
  - Delete all other project indexes after 30 days.

- Run the Curator jobs at midnight every day.
- c. • Delete indexes in the **operations** project older than 30 days.
  - Delete all other project indexes after 10 days.
  - Run the Curator jobs at 10:00 a.m every day.
- d. • Delete indexes in the **logging-devel** project older than 30 days.
  - Delete all other project indexes after 10 days.
  - Run the Curator jobs at midnight every day.

## Solution

Choose the correct answers to the following questions:

1. Which of the following services is the data collector that gather logs from the application nodes?
  - a. Curator
  - b. **Fluentd**
  - c. Kibana
  - d. Elasticsearch
2. Which two storage back ends are recommended for use with Elasticsearch? (Choose two.)
  - a. GlusterFS
  - b. CephFS
  - c. NFS
  - d. CIFS
3. Which of the following objects is used for deploying Fluentd?
  - a. LogConfig
  - b. ConfigMaps
  - c. DeploymentConfig
  - d. **DaemonSet**
4. Which of the following lists describes the Curator configuration file presented below?

```
logging-devel:
 delete:
 days: 10

.operations:
 delete:
 days: 30

.defaults:
 delete:
 days: 30
 runhour: 0
 runminute: 0
```

- a.
  - Delete indexes in the **default** project older than 30 days.
  - Delete all other project indexes after 10 days.
  - Run the Curator jobs at midnight every day.
- b.
  - Delete indexes in the **logging-devel** project older than 10 days.
  - Delete all other project indexes after 30 days.
  - Run the Curator jobs at midnight every day.

- c.
  - Delete indexes in the **operations** project older than 30 days.
  - Delete all other project indexes after 10 days.
  - Run the Curator jobs at 10:00 a.m every day.
- d.
  - Delete indexes in the **logging-devel** project older than 30 days.
  - Delete all other project indexes after 10 days.
  - Run the Curator jobs at midnight every day.

# Enabling Log Aggregation in an HA Cluster

## Objectives

After completing this section, students should be able to:

- Discuss the storage and sizing considerations for EFK.
- Describe the installation procedure for EFK in OpenShift Container Platform.
- Discuss troubleshooting techniques for an EFK stack.

## Introduction to Elasticsearch

Before installing an EFK stack, you must ensure that the environment meets the sizing requirements for the stack, especially for Elasticsearch. Each Elasticsearch instance requires its own individual storage, but an OpenShift Container Platform deployment can only provide volumes shared by all of the pods that comprise the stack. This means that Elasticsearch should not be implemented as a single instance, rather, with multiple instances.

### Sizing Considerations

The following table describes the disk space requirements, assuming the following scenario:

- Application: Apache
- 256 bytes per line
- 60 lines per second per pod
- Raw text format: JSON

| Storage Considerations Overview                                                   |                                                                  |
|-----------------------------------------------------------------------------------|------------------------------------------------------------------|
| Logging Pods                                                                      | Storage Throughput                                               |
| Three Elasticsearch instances, one Kibana, one Curator, and one Fluentd instance. | Six pods total: <b>256 x 60 x 6 x 24 x 60 x 60 = 7.8 GB/day</b>  |
| Three Elasticsearch instances, one Kibana, one Curator, and 11 Fluentd instances. | 16 pods total: <b>256 x 60 x 16 x 24 x 60 x 60 = 21.0 GB/day</b> |

To calculate the total logging throughput and amount of disk space required for aggregated logging, you must have knowledge of the application, and how much logging data it creates.

For example, if one of the applications averages ten lines per second, each with a size of 256 bytes, the following formula determines the application throughput and the disk space requirements:

1. **bytes per line x lines per second** = 2560 bytes per application per second
2. **2560 x number of pods per node (100)** = 256 000 bytes per second per node
3. **256 000 x number of nodes** = Total logging throughput per cluster

### Storage Considerations

Add a persistent volume to each Elasticsearch deployment configuration. On OpenShift Container Platform, this is often achieved through persistent volume claims.

Use SSD drives to optimize throughput. In Red Hat Enterprise Linux 7, the **deadline I/O** scheduler is the default scheduler for all block devices except SATA disks. For SATA disks, the default I/O scheduler is **cfq** (*Completely Fair Queueing*).

The required storage depends on how you optimize the indexes, but a recommended practice is to keep storage usage around 50% and below 70%.

### Storage Configuration

The OpenShift deployer creates an ephemeral deployment, where all of a pod's data is lost upon restart. For production usage, add a persistent storage volume to each Elasticsearch deployment configuration. The **openshift\_hosted\_logging\_storage\_kind** variable configures the storage for the logging environment. This variable must be set to use persistent storage for logging. If the variable is not set, then cluster logging data is stored in an **EmptyDir** volume, which is deleted when the Elasticsearch pod terminates.

The following set of variables defines an NFS back end for the EFK stack:

```
openshift_hosted_logging_storage_kind=nfs①
openshift_hosted_logging_storage_access_modes=['ReadWriteOnce']②
openshift_hosted_logging_storage_host=console.lab.example.com③
openshift_hosted_logging_storage_nfs_directory=/exports④
openshift_hosted_logging_storage_volume_name=logging-es⑤
openshift_hosted_logging_storage_volume_size=10Gi⑥
```

- ① Sets the storage back end to NFS.
- ② Defines the access mode for the volume.
- ③ Defines the NFS server, if not present in the **[nfs]** group of the Ansible inventory file.
- ④ Defines the NFS directory that contains the share.
- ⑤ Defines the NFS share to use.
- ⑥ Defines the size of the volume for Elasticsearch.

There are three options for configuring the storage back end when using the advanced installation:

#### NFS Host Group

With this option, an NFS volume is created during the advanced installation. The volume uses **nfs\_directory/volume\_name/** on the host within the **[nfs]** host group.

```
[OSEv3:vars]
...
openshift_hosted_logging_storage_kind=nfs
openshift_hosted_logging_storage_access_modes=['ReadWriteOnce']
openshift_hosted_logging_storage_nfs_directory=/exports
openshift_hosted_logging_storage_nfs_options='*(rw,root_squash)'
openshift_hosted_logging_storage_volume_name=logging
openshift_hosted_logging_storage_volume_size=10Gi
```

### External NFS host

With this option, the NFS share must exist on the storage host. The remote volume path using the following options would be `nfs.example.com:/exports/logging`.

```
[OSEv3:vars]
...
openshift_hosted_logging_storage_kind=nfs
openshift_hosted_logging_storage_access_modes=['ReadWriteOnce']
openshift_hosted_logging_storage_host=nfs.example.com
openshift_hosted_logging_storage_nfs_directory=/exports
openshift_hosted_logging_storage_volume_name=logging
openshift_hosted_logging_storage_volume_size=10Gi
```

### Dynamic

With this option, OpenShift Container Platform dynamically provisions the volume. You can provision OpenShift Container Platform with dynamically allocated storage when running in a cloud environment:

```
[OSEv3:vars]
...
openshift_hosted_logging_storage_kind=dynamic
```

### Deployment Prerequisites

The following requirements must also be satisfied before you install the EFK stack:

- A router must be present in the cluster. Routers are the ingress point for all external traffic destined for services in the cluster. The Ansible Playbook that installs the logging stack creates a set of routes for public access to the Kibana and Elasticsearch instances.
- A **Persistent Volume** must be declared in the project in which the EFK stack is deployed. The Ansible Playbook creates a **Persistent Volume Claim** for Elasticsearch.
- An OpenShift project, for example **logging**, must be present. If the project does not exist, the Ansible Playbook creates it. Create the project if you need to add a node selector.

### Setting the Ansible Inventory Variable for EFK

OpenShift Container Platform ships with an Ansible Playbook that automatically deploys and configures the logging environment. The playbook is located at `/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster/openshift-logging.yml` and includes several roles that configure Elasticsearch, Kibana, Fluentd, and Curator. To install aggregated logging, run the playbook against the Ansible inventory file used to deploy OpenShift Container Platform.

To install the logging environment, update the inventory file with the necessary variables. Only one variable is required to install the EFK stack, **openshift\_logging\_install\_logging=true**, but many variables exist that you can use to adjust the deployment according to the environment. The following excerpt shows how to configure logging in an offline environment:

```
openshift_logging_install_logging=true
openshift_hosted_logging_deployer_prefix=registry.lab.example.com:5000/openshift3/
```

```

openshift_logging_use_ops=false3
openshift_logging_kibana_hostname=kibana.apps.lab.example.com4
openshift_logging_fluentd_memory_limit='256Mi'5
openshift_logging_es_memory_limit='8Gi'6
openshift_logging_es_allow_external=True7
openshift_logging_es_hostname=elasticsearch.apps.lab.example.com8
openshift_logging_image_version=latest9

openshift_hosted_logging_deployer_version=latest10
openshift_hosted_logging_storage_kind=nfs11
openshift_hosted_logging_storage_access_modes=['ReadWriteOnce']12
openshift_hosted_logging_storage_nfs_directory=/exports13
openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'14
openshift_hosted_logging_storage_volume_name=logging-es15
openshift_hosted_logging_storage_volume_size=5Gi16

```

- <sup>1</sup> Set to **true** to install logging. Set to **false** to uninstall logging.
- <sup>2</sup> The URL of the custom registry for offline deployment.
- <sup>3</sup> Set to **true** to configure a second Elasticsearch cluster and Kibana for operations logs. Fluentd splits logs between the main cluster and a cluster reserved for operations logs. This option deploys secondary Elasticsearch and Kibana instances.
- <sup>4</sup> The external host name for web clients to reach Kibana.
- <sup>5</sup> The memory limit for Fluentd pods.
- <sup>6</sup> The amount of RAM to reserve per Elasticsearch instance, which should be least 512 MB.
- <sup>7</sup> Set to **true** to expose Elasticsearch as a route.
- <sup>8</sup> The external facing host name to use for the route and the TLS server certificate.
- <sup>9</sup> The image version for the logging images to use.
- <sup>10</sup> The image version for the deployer images to use.
- <sup>11</sup> The storage back end to use.
- <sup>12</sup> The volume access mode.
- <sup>13</sup> The name of the NFS share to use for Elasticsearch.
- <sup>14</sup> The storage back end options.
- <sup>15</sup> The name of the NFS volume.
- <sup>16</sup> The size to allocate for Elasticsearch storage.



### Note

Consult the Aggregating Container Logs section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* for a complete list of variables.

### Installing the EFK Stack

The Ansible Playbook deploys all the resources required for the stack. This includes the secrets, the service accounts, and the deployment configurations. The playbook is displayed below:

```

#
This playbook is a preview of upcoming changes for installing
Hosted logging on. See inventory/byo/hosts.*.example for the
currently supported method.
#
- include: initialize_groups.yml

- include: ../../common/openshift-cluster/openshift_logging.yml
 vars:
 openshift_cluster_id: "{{ cluster_id | default('default') }}"
 openshift_debug_level: "{{ debug_level | default(2) }}"

```

To deploy the services in the cluster, run the playbook against the inventory file used for the initial deployment:

```
[root@demo ~]# ansible-playbook -i inventory file \
/usr/share/ansible/openshift-ansible/playbooks/byo/ \
openshift-cluster/openshift-logging.yml
```

The Ansible Playbook waits until the stack is running before deploying the component pods. During the deployment, run the **oc get pods** command to monitor the installation process:

```
[root@demo ~]# oc get pods -w
```

To review the logs and to ensure that they are running successfully, run the following command:

```
[root@demo ~]# oc logs -f pod-name
```

To clean up the entire EFK stack, run the following command:

```
[root@demo ~]# ansible-playbook -i inventory-file \
/usr/share/ansible/openshift-ansible/playbooks/common/ \
openshift-cluster/openshift_logging.yml \
-e openshift_logging_install_logging=False
```

After the execution of the Ansible Playbook, the following resources are available in the **logging** project:

- Three deployment configurations, one for Curator, one for Elasticsearch, and one for Kibana.

| NAME                               | REVISION | DESIRED | CURRENT |   |
|------------------------------------|----------|---------|---------|---|
| dc/logging-curator                 |          | 1       | 1       | 1 |
| dc/logging-es-data-master-185mjk1r | 1        | 1       | 1       |   |
| dc/logging-kibana                  | 2        | 1       | 1       |   |

- Four replication controllers, one for Curator, one for Fluentd, and two for Kibana:

| NAME                                 | DESIRED | CURRENT | READY | AGE |
|--------------------------------------|---------|---------|-------|-----|
| rc/logging-curator-1                 | 1       | 1       | 1     | 4h  |
| rc/logging-es-data-master-185mjk1r-1 | 1       | 1       | 1     | 4h  |
| rc/logging-kibana-1                  | 0       | 0       | 0     | 4h  |
| rc/logging-kibana-2                  | 1       | 1       | 1     | 4h  |

- Two routes, one for Kibana and one for Elasticsearch:

| NAME                  | HOST/PORT                          |                    |             |          |
|-----------------------|------------------------------------|--------------------|-------------|----------|
| routes/logging-es     | elasticsearch.apps.lab.example.com |                    |             |          |
| PATH                  | SERVICES                           | PORT               | TERMINATION | WILDCARD |
| routes/logging-kibana | kibana.apps.lab.example.com        |                    |             |          |
| logging-es            | <all>                              | reencrypt          | None        |          |
| logging-kibana        | <all>                              | reencrypt/Redirect | None        |          |

- Three services, two for Elasticsearch and one for Kibana:

| NAME                   | CLUSTER-IP     | EXTERNAL-IP | PORT(S)  | AGE |
|------------------------|----------------|-------------|----------|-----|
| svc/logging-es         | 172.30.79.164  | <none>      | 9200/TCP | 4h  |
| svc/logging-es-cluster | 172.30.53.36   | <none>      | 9300/TCP | 4h  |
| svc/logging-kibana     | 172.30.136.172 | <none>      | 443/TCP  | 4h  |

- A set of pods for the three services:

| NAME                                       | READY | STATUS  | RESTARTS | AGE |
|--------------------------------------------|-------|---------|----------|-----|
| po/logging-curator-1-rc55z                 | 1/1   | Running | 0        | 4h  |
| po/logging-es-data-master-185mjk1r-1-hzjj6 | 1/1   | Running | 0        | 4h  |
| po/logging-fluentd-3f9cf                   | 1/1   | Running | 0        | 4h  |
| po/logging-fluentd-6sgxg                   | 1/1   | Running | 0        | 4h  |
| po/logging-fluentd-9fbgb                   | 1/1   | Running | 0        | 4h  |
| ...                                        |       |         |          |     |

## Troubleshooting the Logging Environment

If you experience issues with aggregated logging, such as an inability to connect to the Kibana web console, or error messages indicating that Elasticsearch is not reachable, review the following troubleshooting sections. Consult the references for further troubleshooting.

### Login Loop

The OAuth2 proxy on the Kibana console must share a secret with the master host's OAuth2 server. If the secret is not identical on both servers, it can cause a login loop where you are continuously redirected back to the Kibana login page.

To fix this issue, delete the current OAuthClient, and use the **ansible-playbook** command to rerun the **openshift\_logging** role:

```
[root@demo ~]# oc delete oauthclient/kibana-proxy
[root@demo ~]# ansible-playbook -i inventory \
/usr/share/ansible/openshift-ansible/playbooks/byo/ \
openshift-cluster/openshift-logging.yml
```

### 503 Error When Viewing the Console

If you receive a proxy error when viewing the Kibana console, it could be caused by one of two issues:

- Kibana may not be recognizing pods. If Elasticsearch is slow to start, Kibana may time out trying to reach it. Check whether the relevant service has any endpoints:

```
[root@demo ~]# oc describe service logging-kibana
Name: logging-kibana
...

```

```
Endpoints: <none>
```

If any Kibana pods are alive, retrieve the endpoints. If the endpoints are not listed, retrieve the state of the Kibana pods and deployment. You may need to scale the deployment down and back up again.

- The second possible issue may be that the route for accessing the Kibana service is masked. This can happen if you perform a test deployment in one project, then deploy in a different project without completely removing the first deployment.

When multiple routes are sent to the same destination, the default router will only route to the first created destination. Review the problematic route to see if it is defined in multiple places:

```
[root@demo ~]# oc get route --all-namespaces --selector logging-infra=support
```



## References

Further information is available in the Aggregating Container Logs section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at

<https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Aggregating Sizing Guidelines section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at

<https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Enabling Log Aggregation in an HA Cluster

In this exercise, you will deploy the EFK stack in OpenShift Container Platform and connect to the Kibana web console to ensure that the logging platform is successfully installed.

## Outcomes

You should be able to:

- Use the **git** command.
- Update the Ansible inventory file for OpenShift Container Platform with the necessary changes to configure logging in the cluster.
- Execute the Ansible Playbook for the installation of the logging environment.
- Update an OpenShift deployment configuration.
- Review the logging pods deployed in the logging project.
- Log in to the Kibana web console.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab logging-install setup** command. The script installs the required files for this exercise and creates a persistent volume:

```
[student@workstation ~]$ lab logging-install setup
```

## Steps

1. On the **workstation** VM, open a terminal and run the **oc login** command to connect to the cluster, available at <https://console.lab.example.com>.

Use **admin** as the user name and **redhat** as the password. If prompted, agree to use an insecure connection:

```
[student@workstation ~]$ oc login https://console.lab.example.com \
-u admin -p redhat
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

* default
 glusterfs
 kube-public
 kube-system
 logging
 management-infra
 openshift
 openshift-infra
 teststorage
```

```
Using project "default".
```

- Run the **oc get svc** command to retrieve the list of services declared in the **default** project in the cluster. Ensure that a router is listed:

```
[student@workstation ~]$ oc get svc
NAME CLUSTER-IP PORT(S) AGE
... output omitted ...
router 172.30.100.115 80/TCP,443/TCP,1936/TCP 44m
```

- Ensure that the router pods are running. There should be three router pods, one for each infrastructure node. Use the **oc get pods** command with the **-o wide** option to list the nodes on which the pods are running:

```
[student@workstation ~]$ oc get pods -o wide
NAME READY STATUS ... NODE
... output omitted ...
router-2-d2s12 1/1 Running ... infra3.lab.example.com
router-2-szn4r 1/1 Running ... infra1.lab.example.com
router-2-tshj4 1/1 Running ... infra2.lab.example.com
```

- List the persistent volumes. The lab script created a volume called **logging-volume** with a size of 10 Gi. Ensure that the volume is marked as **Available**.

```
[student@workstation ~]$ oc get pv
NAME CAPACITY ACCESSMODES RECLAIMPOLICY STATUS
... output omitted ...
logging-volume 10Gi RWO Retain Available
```

- On the **workstation** VM, use the **ssh** command to log in to the **console** VM as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

- Ensure that the **openshift-install** directory, created in the section called “*Guided Exercise: Using the Advanced Installation Method*” is present in the **/root** directory. If not, run the **git clone** command to retrieve the repository from the **services** VM:

```
[root@console ~]# git clone \
http://services.lab.example.com/openshift-install
```

- Inspect the files from the cloned repository.

Go to the **openshift-install/lab.example.com** directory and list the files:

```
[root@console ~]# cd openshift-install/lab.example.com/
[root@console lab.example.com]# ls
hosts-ucf README.md rootCA.crt rootCA.csr rootCA.key
```

8. Open the **hosts-ucf** file with a text editor and add the necessary variables for the deployment of the EFK stack. Append the following variables to the **[OSEv3:vars]** group.



### Note

You can use the file located at **/root/D0380/labs/logging-install/extravariables-inventory.yaml** to copy and paste the variable into the Ansible inventory file.

- **openshift\_logging\_install\_logging=true**
- **openshift\_hosted\_logging\_deployer\_prefix=registry.lab.example.com:5000/openshift3/**
- **openshift\_logging\_use\_ops=false**
- **openshift\_logging\_kibana\_hostname=kibana.apps.lab.example.com**
- **openshift\_logging\_fluentd\_memory\_limit='128Mi'**
- **openshift\_logging\_es\_memory\_limit='3Gi'**
- **openshift\_logging\_es\_allow\_external=True**
- **openshift\_logging\_es\_hostname=elasticsearch.apps.lab.example.com**
- **openshift\_logging\_image\_version=latest**
- **openshift\_logging\_kibana\_nodeselector="{'region': 'infra'}"**
- **openshift\_logging\_curator\_nodeselector="{'region': 'infra'}"**
- **openshift\_logging\_es\_nodeselector="{'region': 'infra'}"**
- **openshift\_hosted\_logging\_deployer\_version=latest**
- **openshift\_hosted\_logging\_storage\_kind=nfs**
- **openshift\_hosted\_logging\_storage\_access\_modes=['ReadWriteOnce']**
- **openshift\_hosted\_logging\_storage\_nfs\_directory=/exports**
- **openshift\_hosted\_logging\_storage\_nfs\_options='\*(rw,root\_squash)'**
- **openshift\_hosted\_logging\_storage\_volume\_name=logging-es**
- **openshift\_hosted\_logging\_storage\_volume\_size=10Gi**

The variables enable the EFK stack and define the NFS storage for Elasticsearch. The storage is used to store the engine data.

The **[OSEv3:vars]** group should read as follows:

```
[OSEv3:vars]
openshift_deployment_type=openshift-enterprise
```

---

```

deployment_type=openshift-enterprise
openshift_release=v3.6

openshift_master_api_port=443
openshift_master_console_port=443
openshift_portal_net=172.30.0.0/16
osm_cluster_network_cidr=10.128.0.0/14

openshift_master_cluster_method=native
openshift_master_cluster_hostname=console.lab.example.com
openshift_master_cluster_public_hostname=console.lab.example.com
openshift_master_default_subdomain=apps.lab.example.com

openshift_master_identity_providers=[{'name': 'htpasswd_auth', 'login': 'true',
'challenge': 'true', 'kind': 'HTPasswdPasswordIdentityProvider', 'filename': '/etc/
origin/master/htpasswd'}]
openshift_master_htpasswd_users={'admin': '$apr1$4ZbKL261$3eKL/6AQM80941RwTAu611',
'developer': '$apr1$4ZbKL261$3eKL/6AQM80941RwTAu611'}

#openshift_hosted_registry_storage_kind=nfs
#openshift_hosted_registry_storage_access_modes=['ReadWriteOnce']
#openshift_hosted_registry_storage_nfs_directory=/exports
#openshift_hosted_registry_storage_nfs_options='*(rw,root_squash)'
#openshift_hosted_registry_storage_volume_name=registry
#openshift_hosted_registry_storage_volume_size=10Gi
openshift_hosted_registry_storage_kind=glusterfs

openshift_router_selector='region=infra'
openshift_registry_selector='region=infra'

openshift_set_node_ip=True
ansible_ssh_user=root

openshift_disable_check=disk_availability,docker_storage,memory_availability, \
docker_image_availability
openshift_cockpit_deployer_prefix='openshift3/'

#running offline
openshift_docker_additional_registries=registry.lab.example.com:5000
openshift_docker_insecure_registries=registry.lab.example.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com, docker.io

openshift_master_ca_certificate={'certfile': '/root/openshift-install/
lab.example.com/rootCA.crt', 'keyfile': '/root/openshift-install/lab.example.com/
rootCA.key'}

oreg_url=registry.lab.example.com:5000/openshift3/ose-${component}:${version}
openshift_examples_modify_imagestreams=true

#logging
openshift_logging_install_logging=true

openshift_hosted_logging_deployer_prefix=registry.lab.example.com:5000/openshift3/
openshift_logging_use_oss=false
openshift_logging_kibana_hostname=kibana.apps.lab.example.com
openshift_logging_fluentd_memory_limit='128Mi'
openshift_logging_es_memory_limit='3Gi'
openshift_logging_es_allow_external=True
openshift_logging_es_hostname=elasticsearch.apps.lab.example.com
openshift_logging_image_version=latest
openshift_logging_kibana_nodeselector="{'region': 'infra'}"
openshift_logging_curator_nodeselector="{'region': 'infra'}"
openshift_logging_es_nodeselector="{'region': 'infra'}"

```

```
openshift_hosted_logging_deployer_version=latest
openshift_hosted_logging_storage_kind=nfs
openshift_hosted_logging_storage_access_modes=['ReadWriteOnce']
openshift_hosted_logging_storage_nfs_directory=/exports
openshift_hosted_logging_storage_volume_name=logging-es
openshift_hosted_logging_storage_volume_size=10Gi
```

9. Save and exit the file.
10. From **workstation**, open a new terminal and run the **lab logging-install grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab logging-install grade
...
Comparing Entries in [OSEv3:vars]

. Checking openshift_deployment_type..... PASS
. Checking deployment_type..... PASS
. Checking openshift_release..... PASS
. Checking osm_cluster_network_cidr..... PASS
. Checking openshift_portal_net..... PASS
...
```

11. Push the changes to Git.
- 11.1. From the **console** VM, add the updated Ansible inventory file to the Git index.

Go to the **/root/openshift-install** directory and add all the updates to the Git index.

```
[root@console ~]# cd /root/openshift-install
[root@console openshift-install]# git add lab.example.com/*
```

- 11.2. Commit the Ansible inventory file to the Git repository with the comment **Installing logging tools**.

```
[root@console openshift-install]# git commit -m \
'Installing logging tools'
```

- 11.3. Push the **openshift-cluster** project directory to the Git repository.

```
[root@console openshift-install]# git push
...
Counting objects: 10, done.
...
9d01a61..37260b7 master -> master
```

12. Run the **docker-registry-cli** command with the **search logging** option to search for the logging images available at **registry.lab.example.com**. Ensure that the following images are present:

  - **openshift3/logging-deployer**

- **openshift3/logging-kibana**
- **openshift3/logging-curator**
- **openshift3/logging-auth-proxy**
- **openshift3/logging-elasticsearch**
- **openshift3/logging-fluentd**

```
[root@console openshift-install]# docker-registry-cli \
registry.lab.example.com:5000 search logging

1) Name: openshift3/logging-fluentd
Tags: latest

2) Name: openshift3/logging-kibana
Tags: latest

3) Name: openshift3/logging-deployer
Tags: latest

4) Name: openshift3/logging-auth-proxy
Tags: latest

5) Name: openshift3/logging-elasticsearch
Tags: latest

6) Name: openshift3/logging-curator
Tags: latest

6 images found !
```

13. Execute the Ansible Playbook that installs the EFK stack against the inventory file. The playbook is located at **/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster/openshift-logging.yml**

Review the playbook **RECAP** section to ensure that there are no errors.



### Note

If the playbook execution fails:

1. Update the **openshift\_logging\_install\_logging** with a value of **false**.
2. Rerun the Ansible Playbook to remove the EFK stack.
3. Rerun the **lab logging-install setup** command.

```
[root@console openshift-install]# ansible-playbook -i lab.example.com/hosts-ucf \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-cluster/ \
openshift-logging.yml
...
```

```
PLAY RECAP ****
localhost : ok=13 changed=0 unreachable=0 failed=0
master1.lab.example.com : ok=174 changed=23 unreachable=0 failed=0
master2.lab.example.com : ok=14 changed=0 unreachable=0 failed=0
master3.lab.example.com : ok=17 changed=4 unreachable=0 failed=0
```

14. Open a new terminal tab and list the deployment configuration objects in the **logging** project and wait for the **logging-kibana** deployment configuration to appear.

To customize the readiness check needed by the **logging-kibana** image, reimport the **logging-kibana** deployment configuration with the **/home/student/D0380/labs/logging-install/logging-kibana.yaml** template.

To update the **deploymentConfig**, run the **oc replace** command with the **-f** option from the **workstation** VM as the **admin** user.

```
[student@workstation ~]$ oc login -u admin -p redhat
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

* default
 glusterfs
 kube-public
 kube-system
 logging
 management-infra
 openshift
 openshift-infra
 teststorage

Using project "default".
[student@workstation ~]$ oc project logging
Now using project "logging" on server "https://console.lab.example.com:443".
[student@workstation ~]$ oc replace -f \
/home/student/D0380/labs/logging-install/logging-kibana.yaml
deploymentconfig "logging-kibana" replaced
```

15. Review the pods deployed in the logging project to ensure that all the pods are marked as **Running**. Notice the **logging-kibana-2-3hz1q**, comprised of two containers and the Kibana pod from the first deployment configuration in the **terminating** state.

| NAME                                    | READY | STATUS             | RESTARTS | AGE |
|-----------------------------------------|-------|--------------------|----------|-----|
| logging-curator-1-rc55z                 | 1/1   | Running            | 0        | 1m  |
| logging-es-data-master-185mjk1r-1-hzjj6 | 1/1   | Running            | 0        | 2m  |
| logging-fluentd-3f9cf                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-6sgxg                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-9fbgb                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-f4dck                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-gnwkj                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-kbp01                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-lrgt1                   | 1/1   | Running            | 0        | 1m  |
| logging-fluentd-zc62r                   | 1/1   | Running            | 0        | 1m  |
| logging-kibana-1-j01kb                  | 1/2   | <b>terminating</b> | 0        | 5M  |
| logging-kibana-2-3hz1q                  | 2/2   | Running            | 0        | 40s |

- 
16. Run the **oc logs** command against the **kibana** container in the **logging-kibana-2** pod. The latest message should read **Status changed from yellow to green - Kibana index ready**.

```
[student@workstation ~]$ oc logs -f logging-kibana-2-3hz1q -c kibana
...
{"type":"log","@timestamp":"2017-10-26T17:03:49Z",\
"tags":["listening","info"],"pid":15,"message":"Server running at
http://0.0.0.0:5601"}
{"type":"log","@timestamp":"2017-10-26T17:03:49Z",\
"tags":["status","plugin:elasticsearch@1.0.0","info"],\
"pid":15,"state":"green","message":"Status changed from yellow to green - Kibana
index ready","prevState":"yellow","prevMsg":"Waiting for Elasticsearch"}
```

17. From the **workstation** VM, open a web browser and access the Kibana web console at **kibana.apps.lab.example.com**. Use **admin** as the user name and **redhat** as the password.

The page should display the **Discover** page with a column chart. If the column chart is not rendered, click the list from the left column and choose the **project.logging** application.

This concludes the guided exercise.

# Lab: Enabling Log Aggregation

In this lab, you will review the installation of aggregated logging in OpenShift Container Platform and use the Kibana web console to manage searches and widgets.

## Outcomes

You should be able to:

- Review the environment configuration for aggregated logging.
- Review the route for Kibana.
- Use the Kibana web console to review logs and create widgets.

## Before you begin

The guided exercise from the guided exercise *Enabling Log Aggregation in an HA Cluster* must have been completed. If not, complete the guided exercise before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab logging-review setup** command. The script ensures that aggregated logging is deployed, and that the cluster and Kibana are reachable:

```
[student@workstation ~]$ lab logging-review setup
```

## Steps

1. From the **workstation** VM, log in to the OpenShift cluster as the **admin** user, using **redhat** as the password.

From the **logging** project, review the deployment of the EFK stack in the environment by listing the persistent volume claims, the replication controllers, and the services. Ensure that the following resources are present:

- One persistent volume bound to the **logging-es-0** persistent volume
  - Four replication controllers for aggregated logging
  - Three services for aggregated logging
2. Ensure that there is a route for the Kibana service, available at **https://kibana.apps.lab.example.com**. Use the **curl** command with the **-v** and **--insecure** options to access the URL. Locate the output that reads:

```
X-Powered-By: Express
Location: https://master1.lab.example.com:443/oauth/authorize?response_type=code& \
redirect_uri=https%3A%2Fkibana.apps.lab.example.com%2Fauth%2Fopenshift \
%2Fcallback& \
scope=user%3Ainfo%20user%3Acheck-access%20user%3Alist-projects& \
client_id=kibana-proxy
```

This line indicates that the request is redirected to the authentication proxy before redirecting the user to the Kibana console.

- 
3. From **workstation**, log in to the Kibana web console. Use **admin** as the user name and **redhat** as the password. Ensure that the **Discover** page shows some entries in the data table.
  4. From the **Discover** page, select the **project.logging.*UUID*** project.  
Add the **Hostname** search field as a filter for any host, for example **master1.lab.example.com**, and save the filter as **master1 hostname**.
  5. From the **Dashboard** page, create a new widget from the **My hostname** saved search. Collapse the first entry within the widget and inspect the contents.

This concludes the lab.

## Solution

In this lab, you will review the installation of aggregated logging in OpenShift Container Platform and use the Kibana web console to manage searches and widgets.

### Outcomes

You should be able to:

- Review the environment configuration for aggregated logging.
- Review the route for Kibana.
- Use the Kibana web console to review logs and create widgets.

### Before you begin

The guided exercise from the guided exercise *Enabling Log Aggregation in an HA Cluster* must have been completed. If not, complete the guided exercise before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab logging-review setup** command. The script ensures that aggregated logging is deployed, and that the cluster and Kibana are reachable:

```
[student@workstation ~]$ lab logging-review setup
```

### Steps

1. From the **workstation** VM, log in to the OpenShift cluster as the **admin** user, using **redhat** as the password.

From the **logging** project, review the deployment of the EFK stack in the environment by listing the persistent volume claims, the replication controllers, and the services. Ensure that the following resources are present:

- One persistent volume bound to the **logging-es-0** persistent volume
- Four replication controllers for aggregated logging
- Three services for aggregated logging

- 1.1. From the **workstation** VM, open a new terminal and log in to the OpenShift cluster as the **admin** user, using **redhat** as the password:

```
[student@workstation ~]$ oc login \
https://console.lab.example.com \
-u admin -p redhat
Login successful.

You have access to the following projects and can switch between them with 'oc
project <projectname>':

 default
 glusterfs
 kube-public
 kube-system
* logging
 management-infra
 openshift
```

```
openshift-infra
teststorage
```

Using project "logging" on server "https://console:443".

- 1.2. If necessary, use the **oc project** command to change to the **logging** project:

```
[student@workstation ~]$ oc project logging
Now using project "logging" on server "https://console.lab.example.com:443".
```

- 1.3. List the persistent volumes in the **logging** project to ensure that the volume called **logging-volume** is present and that it is bound to the **logging-es-0** persistent volume claim:

```
[student@workstation ~]$ oc get pv
NAME CAPACITY ACCESSMODES RECLAIMPOLICY
logging-volume 10G RWO Retain
...
STATUS CLAIM STORAGECLASS REASON AGE
Bound logging/logging-es-0
...
...
```

- 1.4. List the replication controllers in the **logging** project. Ensure that there are four replication controllers:

```
[student@workstation ~]$ oc get rc
NAME DESIRED CURRENT READY AGE
logging-curator-1 1 1 1 1d
logging-es-data-master-bvfaqk8o-1 1 1 1 1d
logging-kibana-1 0 0 0 1d
logging-kibana-2 1 1 1 1d
```

- 1.5. Retrieve the list of the services. Ensure that there are three services:

```
[student@workstation ~]$ oc get svc
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
logging-es 172.30.115.33 <none> 9200/TCP 1d
logging-es-cluster 172.30.254.54 <none> 9300/TCP 1d
logging-kibana 172.30.168.183 <none> 443/TCP 1d
```

2. Ensure that there is a route for the Kibana service, available at **https://kibana.apps.lab.example.com**. Use the **curl** command with the **-v** and **--insecure** options to access the URL. Locate the output that reads:

```
X-Powered-By: Express
Location: https://master1.lab.example.com:443/oauth/authorize?response_type=code& \
redirect_uri=https%3A%2F%2Fkibana.apps.lab.example.com%2Fauth%2Fopenshift \
%2Fcallback& \
scope=user%3Ainfo%20user%3Acheck-access%20user%3Alist-projects& \
client_id=kibana-proxy
```

This line indicates that the request is redirected to the authentication proxy before redirecting the user to the Kibana console.

- 2.1. As the **admin** user in the **logging** project, run the **oc get routes** command to list the routes in the project:

```
[student@workstation ~]$ oc get routes
NAME HOST/PORT
logging-es elasticsearch.apps.lab.example.com
logging-kibana kibana.apps.lab.example.com
PATH SERVICES PORT TERMINATION WILDCARD
 logging-es <all> reencrypt None
 logging-kibana <all> reencrypt/Redirect None
```

- 2.2. Use the **curl** command with the **-v** and **--insecure** options to ensure that the server is reachable. Locate the line that shows the redirection to the **master1** VM for authentication purposes:

```
[student@workstation ~]$ curl -v --insecure https://kibana.apps.lab.example.com/
...
Location: https://master1.lab.example.com:443/oauth/authorize?
response_type=code& \
redirect_uri=https%3A%2F%2Fkibana.apps.lab.example.com%2Fauth%2Fopenshift
%2Fcallback& \
scope=user%3Ainfo%20user%3Acheck-access%20user%3Alist-projects&
client_id=kibana-proxy
...
```

3. From **workstation**, log in to the Kibana web console. Use **admin** as the user name and **redhat** as the password. Ensure that the **Discover** page shows some entries in the data table.
  - 3.1. From **workstation**, open Firefox and go to **https://kibana.apps.lab.example.com**. When prompted, accept the self-signed security certificate. After the redirection to **master1.lab.example.com**, log in to Kibana using **admin** as the user name and **redhat** as the password.
  - 3.2. The default page should be the **Discover** page. Ensure that there is a column chart and entries underneath, sorted by time.
4. From the **Discover** page, select the **project.logging.*UUID*** project. Add the **Hostname** search field as a filter for any host, for example **master1.lab.example.com**, and save the filter as **master1 hostname**.
  - 4.1. Click the **.operations.\*** drop-down below the search bar and select the **project.logging.*UUID*** project.
  - 4.2. From the **Available Fields** entries, locate the **hostname** field and click **Add** next to it to add the host name as a filter. It is now available in the **Selected Fields** column.
  - 4.3. To filter the results with all discovered host names, click the **hostname** filter in the **Selected Fields** section that you added. Click the magnifier icon with a plus sign (+) next to any host, for example, **master1.lab.example.com** to filter the results with this host name.

Notice the new entry under the search bar, which reads **hostname: "master1.lab.example.com"**.

- 4.4. Click the **Save Search** icon next to the search bar, represented as a diskette. Use **My hostname** as the name of the search and click **Save**.
5. From the **Dashboard** page, create a new widget from the **My hostname** saved search. Collapse the first entry within the widget and inspect the contents.
  - 5.1. Click **Dashboard** to access the Kibana dashboards. On the dashboards page, click **+** below the text that reads **Ready to get started?** to add a new dashboard.

A new frame comprised of two tabs appears. Click the **Searches** tab to list all saved searches, and then click the **My hostname** entry.

Notice the new widget that appears below the search field, which lists a table that filters on the host that you selected, for example, **master1.lab.example.com**.
  - 5.2. From the widget, click the arrow for the first entry in the table. The output should display a frame with two tabs, **Table** and **JSON**.



### Note

To resize the widget, drag the bottom-right corner.

This concludes the lab.

# Summary

In this chapter, you learned:

- The EFK stack is comprised of the following components:
  - *Elasticsearch*, an open source search engine and object store that provides a distributed RESTful API for logs.
  - *Fluentd*, a data collector project that gathers logs from application nodes and sends them to the Elasticsearch service.
  - *Kibana*, a web interface for Elasticsearch.
- Kibana is the web interface that reads log entries from the Elasticsearch database. It can create visualization graphs, charts, time tables, and reports, using time-based and non-time-based events. It also includes a console to run advanced requests.
- Curator is the service that removes old indexes from Elasticsearch on a per-project basis. It allows you to configure maintenance schedules for Elasticsearch.
- A persistent volume should be added to each Elasticsearch deployment configuration. On OpenShift Container Platform, this is often achieved through persistent volume claims.
- The following requirements must also be satisfied before you install the EFK stack:
  - A router must be present in the cluster. Routers are the ingress point for all external traffic destined for services in the cluster. The Ansible Playbook that installs the logging stack creates a set of routes for public access to the Kibana and Elasticsearch instances.
  - A persistent volume must be declared in the project in which the EFK stack is deployed. The Ansible Playbook creates a persistent volume claim for Elasticsearch.
  - An OpenShift project must be present, for example, **logging**. If the project does not exist, the Ansible Playbook creates it.



## CHAPTER 7

# MAINTAINING AN OPENSHIFT CLUSTER

| Overview          |                                                                                                                                                                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Goal</b>       | Perform maintenance activities on an OpenShift cluster.                                                                                                                                                                                                                                         |
| <b>Objectives</b> | <ul style="list-style-type: none"><li>Diagnose the health of the cluster using the diagnostic tool.</li><li>Back up and restore key data stores in an OpenShift cluster.</li><li>Describe and perform clean-up activities on a cluster.</li><li>Add and remove cluster nodes.</li></ul>         |
| <b>Sections</b>   | <ul style="list-style-type: none"><li>Performing Diagnostic Checks (and Guided Exercise)</li><li>Backing up and Restoring Key Data Stores (and Guided Exercise)</li><li>Cleaning up the Cluster (and Guided Exercise)</li><li>Adding and Removing Cluster Nodes (and Guided Exercise)</li></ul> |
| <b>Quiz</b>       | Maintaining an OpenShift Cluster                                                                                                                                                                                                                                                                |

# Performing Diagnostics Checks

## Objective

Diagnose the health of the cluster using the diagnostics tool.

## OpenShift Diagnostics

OpenShift Container Platform is deployed as code built from source, VM image, a container image, or as enterprise RPMs. To assist the OpenShift architect, a diagnostics tool was added to the OpenShift binary. Use the **oc adm diagnostics** command to diagnose error conditions in the host or cluster. The following list identifies the checks run by the **oc adm diagnostics** command on a host or cluster:

- Verifies that the default router and registry are correctly configured and running.
- Checks **ClusterRoles** and **ClusterRoleBindings** for consistency with the base policy.
- Checks that all of the client configuration contexts can be connected to and are valid.
- Checks that the pods have SDN connectivity.
- Checks that SkyDNS, a DNS server that queries for services in OpenShift, is working properly.
- Validates the node and master configuration on the host.
- Checks that nodes are running and available.
- Checks host logs for known errors.
- Checks that **systemd** units are configured as expected for the host.

### Diagnostics Tool

The **oc adm diagnostics** command checks for the standard configuration files for clients, masters, and nodes for problems. Nonstandard locations are specified with the **--config**, **--master-config**, and **--node-config** options, respectively. The default locations of standard configuration files are shown below:

Master

**/etc/origin/master/master-config.yaml**

Client

**~/ kube/config-file**

Node

**/etc/origin/node/node-config.yaml**

### Master and Node Diagnostics

Running master and node diagnostics against a cluster deployed by Ansible provides the following benefits:

- The configuration files used for the master and node are based on the standard location.
- Services are managed by **systemd** units.

- All components use **journald** for logging.

The log diagnostics tool requires that systemd units and log entries for journald be configured. If your deployment uses a different type of logging facility, the log diagnostic checks do not work properly and are skipped.

- To use the diagnostics tool to run all available diagnostics, and to skip any that do not apply, run the following command:

```
[root@master-demo ~]# oc adm diagnostics
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
Info: Using context for cluster-admin access: 'default/console-lab-example-com:443/
system:admin'
[Note] Performing systemd discovery
...output truncated...
```

- To use the diagnostics tool when the configuration files are in nonstandard locations, run the following command:

```
[root@master-demo ~]# oc adm diagnostics --master-config=path-to-master-config-file \
--node-config=path-to-node-config-file
```

- To run one or more specific diagnostics, append their names to the command: For example, to check that cluster roles are present and that they contain the expected permissions according to base policy, run the following command:

```
[root@master-demo ~]# oc adm diagnostics ClusterRoles
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
Info: Using context for cluster-admin access: 'default/console-lab-example-com:443/
system:admin'

[Note] Running diagnostic: ClusterRoles
 Description: Check that the default ClusterRoles are present and contain the
 expected permissions.

[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Completed with no errors or warnings seen.
```

- In a disconnected environment, use the **--images** option to specify the location of the image template used for the **DiagnosticPod** check. For example, to run diagnostics using the **ose-deployer** image template for the **DiagnosticPod** check, run the following command:

```
[root@master-demo ~]# oc adm diagnostics \
--images=registry.lab.example.com:5000/openshift3/ose-deployer
```



## Note

As of this writing, there is a bug that prevents the images from being pulled correctly to run the **DiagnosticPod** test. This bug is fixed in OpenShift Container Platform 3.7. See 1481147 [[https://bugzilla.redhat.com/show\\_bug.cgi?id=1481147](https://bugzilla.redhat.com/show_bug.cgi?id=1481147)] for more details.

| <b>Some Important Diagnostics Checks</b> |                                                                                                                                                                                                                                                   |
|------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Diagnostic Name</b>                   | <b>Purpose</b>                                                                                                                                                                                                                                    |
| <b>AggregatedLogging</b>                 | Check the aggregated logging integration for proper configuration and operation.                                                                                                                                                                  |
| <b>AnalyzeLogs</b>                       | Check the <b>systemd</b> service logs for problems.                                                                                                                                                                                               |
| <b>ClusterRegistry</b>                   | Check that the cluster has a working Docker registry for builds and image streams.                                                                                                                                                                |
| <b>ClusterRoleBindings</b>               | Check that the default cluster role bindings are present and contain the expected subjects according to base policy.                                                                                                                              |
| <b>ClusterRoles</b>                      | Check that cluster roles are present and contain the expected permissions according to base policy.                                                                                                                                               |
| <b>ClusterRouter</b>                     | Check for a working default router in the cluster.                                                                                                                                                                                                |
| <b>ConfigContexts</b>                    | Check that each context in the client configuration is complete and has connectivity to its API server.                                                                                                                                           |
| <b>DiagnosticPod</b>                     | Creates a pod that runs diagnostics from an application standpoint, which checks that DNS within the pod is working as expected and the credentials for the default service account authenticate correctly to the master API.                     |
| <b>EtcReadWriteVolume</b>                | Check the volume of writes against etcd for a time period and classify them by operation and key. This diagnostic only runs if specifically requested, because it does not run as quickly as other diagnostics and can increase the load on etcd. |
| <b>MasterConfigCheck</b>                 | Check this host's master configuration file for problems.                                                                                                                                                                                         |
| <b>MasterNode</b>                        | Check that the master running on this host is also running a node to verify that it is a member of the cluster SDN.                                                                                                                               |
| <b>NodeConfigCheck</b>                   | Checks this host's node configuration file for problems.                                                                                                                                                                                          |
| <b>NodeDefinitions</b>                   | Check that the nodes defined in the master API are ready and can schedule pods.                                                                                                                                                                   |
| <b>RouteCertificateValidation</b>        | Check all route certificates for those that might be rejected by extended validation.                                                                                                                                                             |
| <b>ServiceExternalIPs</b>                | Check for existing services that specify external IPs, which are disallowed according to master configuration.                                                                                                                                    |

### Some Important Diagnostics Checks

|                   |                                                                                                                                              |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------|
| <b>UnitStatus</b> | Check systemd status for units on this host related to OpenShift Container Platform. Does not require a configuration file to check against. |
|-------------------|----------------------------------------------------------------------------------------------------------------------------------------------|

#### Diagnostic Output

The output from `oc adm diagnostics` provides information about the result of the diagnostic checks. The output information is prefixed with either note, warning, info, or error. For example, the following command runs the **MasterConfigCheck** check on the **ose-docker-registry** service.

```
[root@master-demo ~]# oc adm diagnostics MasterConfigCheck
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
[Note] Performing systemd discovery
①
ERROR: [DS1004 from controller openshift/origin/pkg/diagnostics/systemd/locate_units.go]
 Unable to run `systemctl show origin-master`: exit status 1
 Cannot analyze systemd units.
[Note] Running diagnostic: MasterConfigCheck
 Description: Check the master config file
②
WARN: [DH0005 from diagnostic MasterConfigCheck@openshift/origin/pkg/diagnostics/host/check_master_config.go:52]
 Validation of master config file '/etc/origin/master/master-config.yaml' warned:
 assetConfig.loggingPublicURL: Invalid value: "": required to view aggregated
 container logs in the console
 assetConfig.metricsPublicURL: Invalid value: "": required to view cluster metrics
 in the console

[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Warnings seen: 1
```

- ① The error indicates that during the systemd discovery the **origin-master** service was not found. This type of error can be ignored because we are not using an OpenShift Origin deployment.
- ② The warning indicates that **master-config.yaml** is not configured properly if the desired result is to view aggregated container logs and cluster metrics in the console.

#### Ansible-based Health Checks

You can run additional health checks, available through the Ansible-based tooling used to manage and configure OpenShift Container Platform. The tool can report common deployment problems and uses either the **ansible-playbook** command or the containerized version of *openshift-ansible*.

Although not exhaustive, the following health checks are a set of diagnostic tasks that are meant to be run against the Ansible inventory file for a deployed OpenShift Container Platform cluster, using the provided **health.yml** file.

| <b>Ansible-based Health Checks</b> |                                                                                                                                                             |
|------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Check</b>                       | <b>Purpose</b>                                                                                                                                              |
| <b>etcd_imagedata_size</b>         | Measures the total size of OpenShift Container Platform image data in an Etcd cluster. The check fails if the calculated size exceeds a user-defined limit. |

| Ansible-based Health Checks |                                                                                                                                                                                                                                                            |
|-----------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
|                             | If no limit is specified, the check fails if the size of image data amounts to 50% or more of the currently used space in the Etcd cluster.                                                                                                                |
| <b>etcd_traffic</b>         | Detects unusual traffic on an Etcd host. Fails if a <b>journalctl</b> log entry with an Etcd <b>sync duration warning</b> is found.                                                                                                                        |
| <b>docker_storage</b>       | Only runs on hosts that depend on the <b>dockerd</b> daemon. Ensures that Docker's total usage does not exceed a user-defined limit.<br><br>If no user-defined limit is set, Docker's maximum usage threshold defaults to 90% of the total size available. |
| <b>logging_index_time</b>   | Detects higher than normal time delays between log creation and log aggregation by Elasticsearch in a logging stack deployment. Fails if a new log entry cannot be queried through Elasticsearch within a set timeout. Only runs if logging is enabled.    |

You can either run these tests via the **ansible-playbook** command, or via the Docker CLI.

To run the health checks using the **ansible-playbook** command, specify the cluster's inventory file and run the **health.yml** Ansible Playbook:

```
[root@demo ~]# ansible-playbook -i inventory_file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-checks/health.yml
```

Use the **-e** option to set variables on the command line:

```
[root@demo ~]# ansible-playbook -i inventory_file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-checks/health.yml \
-e openshift_check_logging_index_timeout_seconds=45 \
-e etcd_max_image_data_size_bytes=40000000000
```

To run the health checks in a Docker container, use the **docker** command and specify both the cluster's inventory file and the **health.yml** playbook. The command requires that the user has the permission to run privileged containers.

```
[user@demo ~]# docker run -u $(id -u) \①
-v $HOME/.ssh/id_rsa:/opt/app-root/src/.ssh/id_rsa:Z,ro \②
-v /etc/ansible/hosts:/tmp/inventory:ro \③
-e INVENTORY_FILE=/tmp/inventory \
-e PLAYBOOK_FILE=playbooks/byo/openshift-checks/health.yml \④
-e OPTS="-v -e openshift_check_logging_index_timeout_seconds=45 \
-e etcd_max_image_data_size_bytes=40000000000" \⑤
-e openshift3/ose-ansible
```

- ① Makes the container run with the same UID as the current user, which is required for permissions so that the SSH key can be read inside the container.

- ❷ Mounts SSH keys as a volume under `/opt/app-root/src/.ssh` under normal usage when running the container as a non-root user.
- ❸ Changes `/etc/ansible/hosts` to the location of the cluster's inventory file, if different. This file is bind-mounted to the `/tmp/inventory` directory, which is used according to the `INVENTORY_FILE` environment variable in the container.
- ❹ The `PLAYBOOK_FILE` environment variable is set to the location of the `health.yml` Ansible Playbook relative to `/usr/share/ansible/openshift-ansible` inside the container.
- ❺ Sets any variables desired for a single run with the `-e key=value` format.

## Demonstration: Performing Diagnostic Checks

1. Log in to the `workstation` VM as `student` using `student` as the password.

Run the `demo diagnostics setup` command, which ensures that the OpenShift Container Platform cluster is correctly configured for this exercise:

```
[student@workstation ~]$ demo diagnostics setup
```

2. On the `master1` VM, as the `root` user, run the `oc adm diagnostics` command and review the output:

```
[root@master1 ~]# oc adm diagnostics
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
Info: Using context for cluster-admin access: 'default/console-lab-example-com:443/
system:admin'
[Note] Performing systemd discovery
... output omitted...

[Note] Running diagnostic: DiagnosticPod
 Description: Create a pod to run diagnostics from the application standpoint

WARN: [DCLi2006 from diagnostic DiagnosticPod@openshift/origin/pkg/diagnostics/
client/run_diagnostics_pod.go:135]
 Timed out preparing diagnostic pod logs for streaming, so this diagnostic
 cannot run.
 It is likely that the image 'registry.access.redhat.com/openshift3/ose-
 deployer:v3.6.173.0.49' was not pulled and running yet.
 Last error: (*errors.StatusError[2]) container "pod-diagnostics" in pod "pod-
 diagnostic-test-8tn79" is waiting to start: trying and failing to pull image

[Note] Running diagnostic: NetworkCheck
 Description: Create a pod on all schedulable nodes and run network
 diagnostics from the application standpoint

ERROR: [DNet2005 from diagnostic NetworkCheck@openshift/origin/pkg/diagnostics/
network/run_pod.go:119]
❶ Setting up test environment for network diagnostics failed: Failed to run network
diags test pod and service: [timed out waiting for the condition, timed out waiting
for the condition, Failed to run network diags test pods, failed: 20, total: 20]

... output omitted...

[Note] Running diagnostic: ClusterRegistry
```

```

Description: Check that there is a working Docker registry

ERROR: [DClu1019 from diagnostic ClusterRegistry@openshift/origin/pkg/diagnostics/
cluster/registry.go:343]
② Diagnostics created a test ImageStream and compared the registry IP
it received to the registry IP available via the docker-registry service.

docker-registry : 172.30.138.205:5000
ImageStream registry : docker-registry.default.svc:5000

They do not match, which probably means that an administrator re-created
the docker-registry service but the master has cached the old service
IP address. Builds or deployments that use ImageStreams with the wrong
docker-registry IP will fail under this condition.

To resolve this issue, restarting the master (to clear the cache) should
be sufficient. Existing ImageStreams may need to be re-created.

... output omitted...

[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Warnings seen: 6
[Note] Errors seen: 2

```

- ① Currently, [https://bugzilla.redhat.com/show\\_bug.cgi?id=1481147](https://bugzilla.redhat.com/show_bug.cgi?id=1481147) prevents the network diagnostics pod and test pod images from using a deployed OpenShift version/tag that is not the latest and requires downloading another same image with latest tag.
- ② Currently, [https://bugzilla.redhat.com/show\\_bug.cgi?id=1488059](https://bugzilla.redhat.com/show_bug.cgi?id=1488059) causes the **ClusterRegistry** diagnostic to check the registry given to image streams by default with the known registry service. It compares the IP, but with version 3.6 the image stream now gets a cluster host name for the registry instead of an IP. This results in the diagnostic reporting a false error condition because the IP is not the same as the host name.

This concludes the demonstration.

## References

Further information is available in the *Diagnostics Tool* section of the *OpenShift Container Platform Cluster Administration* guide for OpenShift Container Platform 3.6 available at

<https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Performing Diagnostics Checks

In this exercise, you will use the diagnostics tool to diagnose the health of the cluster.

## Outcomes

You should be able to use the diagnostic tool to diagnose the health of the master and node configuration file.

### Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab diagnostics setup** command. The script ensures that the OpenShift Container Platform cluster is correctly configured for this exercise:

```
[student@workstation ~]$ lab diagnostics setup
```

## Steps

1. Log in to the **master1** VM as the **root** user.

Use the **ssh** command to log in to the **master1** VM as the **root** user:

```
[student@workstation ~]$ ssh root@master1
```

2. Verify the configuration of the node configuration file **/etc/origin/node/node-config.yaml**.

- 2.1. On the **master1** VM, run the **oc adm diagnostics** command with the **NodeConfigCheck** option.

```
[root@master1 ~]# oc adm diagnostics NodeConfigCheck \
--node-config='/etc/origin/node/node-config.yaml'
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
[Note] Performing systemd discovery

[Note] Running diagnostic: NodeConfigCheck
 Description: Check the node config file

Info: Found a node config file: /etc/origin/node/node-config.yaml

ERROR: [DH1004 from diagnostic NodeConfigCheck@openshift/origin/pkg/diagnostics/
host/check_node_config.go:50]
 Validation of node config file '/etc/origin/node/node-config.yaml'
failed:
 servingInfo.certFile: Invalid value: "/etc/origin/master/server.crt":
could not read file: stat /etc/origin/master/server.crt: no such file or
directory
 servingInfo.keyFile: Invalid value: "/etc/origin/master/server.key":
could not read file: stat /etc/origin/master/server.key: no such file or
directory

[Note] Summary of diagnostics execution (version v3.6.173.0.49):
```

```
[Note] Errors seen: 1
```

3. Correct the misconfiguration of `/etc/origin/node/node-config.yaml` based on the output of the previous `oc adm diagnostics` command.

- 3.1. Use a text editor to correct the `certfile`, `clientCA`, and `keyfile` values misconfigured in the `/etc/origin/node/node-config.yaml` file:

```
...output omitted...
certFile: /etc/origin/node/server.crt
clientCA: /etc/origin/node/ca.crt
keyFile: /etc/origin/node/server.key
...output omitted...
```

4. Verify that the `/etc/origin/node/node-config.yaml` file is correctly configured.

- 4.1. On the `master1` VM, run the `oc adm diagnostics` command appending the diagnostic check `NodeConfigCheck` option.

```
[root@master1 ~]# oc adm diagnostics NodeConfigCheck
--node-config='/etc/origin/node/node-config.yaml'
[Note] Determining if client configuration exists for client/cluster diagnostics
[Info] Successfully read a client config file at '/root/.kube/config'
[Note] Performing systemd discovery

[Note] Running diagnostic: NodeConfigCheck
Description: Check the node config file

[Info] Found a node config file: /etc/origin/node/node-config.yaml
[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Completed with no errors or warnings seen.
```

5. Run the `oc adm diagnostics` command to fix the wrong configuration of the master configuration file.

- 5.1. On the `master1` VM, run the `oc adm diagnostics` command appending the diagnostic check `MasterConfigCheck`.

```
[root@master1 ~]# oc adm diagnostics MasterConfigCheck \
--master-config='/etc/origin/master/master-config.yaml'
[Note] Determining if client configuration exists for client/cluster diagnostics
[Info] Successfully read a client config file at '/root/.kube/config'
[Note] Performing systemd discovery

[Note] Running diagnostic: MasterConfigCheck
Description: Check the master config file

ERROR: [DH0004 from diagnostic MasterConfigCheck@openshift/origin/pkg/
diagnostics/host/check_master_config.go:45]
 Validation of master config file '/etc/origin/master/master-config.yaml'
failed:
 kubeletClientInfo.ca: Invalid value: "/etc/origin/master/bundle.crt":
 could not read file: stat /etc/origin/master/bundle.crt: no such file or
 directory
 oauthConfig.masterCA: Invalid value: "/etc/origin/master/bundle.crt":
 could not read file: stat /etc/origin/master/bundle.crt: no such file or
 directory
```

```

 serviceAccountConfig.masterCA: Invalid value: "/etc/origin/master/
bundle.crt": could not read file: stat /etc/origin/master/bundle.crt: no such
file or directory
 servingInfo.clientCA: Invalid value: "/etc/origin/master/bundle.crt":
could not read file: stat /etc/origin/master/bundle.crt: no such file or
directory

WARN: [DH0005 from diagnostic MasterConfigCheck@openshift/origin/pkg/
diagnostics/host/check_master_config.go:52]
 Validation of master config file '/etc/origin/master/master-config.yaml'
warned:
 assetConfig.metricsPublicURL: Invalid value: "": required to view cluster
metrics in the console
 auditConfig.auditFilePath: Required value: audit can not be logged to a
separate file

[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Warnings seen: 1
[Note] Errors seen: 1

```

The error found by the diagnostic tells you that the **/etc/origin/master/master-config.yaml** file has an invalid value for the **masterCA**.

6. Correct the misconfiguration of **/etc/origin/master/master-config.yaml** based on the output of the previous **oc adm diagnostics** command.
  - 6.1. Use a text editor to correct the misconfiguration of **/etc/origin/master/master-config.yaml**. There are two **masterCA** and two **ca** entries that are incorrectly set (they are configured as **bundle.crt** instead of **ca-bundle.crt**.)

```

...output omitted...
masterCA: ca-bundle.crt
...output omitted...

```



## Note

All four entries must be updated.

7. Rerun the diagnostics tool to verify the master configuration file is correctly configured.
  - 7.1. On the **master1** VM, run the **oc adm diagnostics** command appending the diagnostic check **MasterConfigCheck**.

```

[root@master1 ~]# oc adm diagnostics MasterConfigCheck \
--master-config='/etc/origin/master/master-config.yaml'
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
[Note] Performing systemd discovery

[Note] Running diagnostic: MasterConfigCheck
 Description: Check the master config file

WARN: [DH0005 from diagnostic MasterConfigCheck@openshift/origin/pkg/
diagnostics/host/check_master_config.go:52]
 Validation of master config file '/etc/origin/master/master-config.yaml'
warned:

```

```

assetConfig.loggingPublicURL: Invalid value: "": required to view
aggregated container logs in the console
assetConfig.metricsPublicURL: Invalid value: "": required to view cluster
metrics in the console
auditConfig.auditFilePath: Required value: audit can not be logged to a
separate file

[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Warnings seen: 1

```

- Verify that the role bindings are present and contain the expected subjects.

Verify that the default **ClusterRoleBindings** is present and contain the expected subjects. Use the **oc adm diagnostics** command with the **ClusterRoleBindings** check in the master configuration file.

- On the **master1** VM, run the **oc adm diagnostics** command appending the diagnostic check **ClusterRoleBindings**.

```

[root@master1 ~]# oc adm diagnostics ClusterRoleBindings \
--master-config='/etc/origin/master/master-config.yaml'
[Note] Determining if client configuration exists for client/cluster diagnostics
Info: Successfully read a client config file at '/root/.kube/config'
Info: Using context for cluster-admin access: 'default/console-lab-example-
com:443/system:admin'

[Note] Running diagnostic: ClusterRoleBindings
Description: Check that the default ClusterRoleBindings are present and
contain the expected subjects

Info: clusterrolebinding/cluster-readers has more subjects than expected.

 Use the `oadm policy reconcile-cluster-role-bindings` command to update
 the role binding to remove extra subjects.

Info: clusterrolebinding/cluster-readers has extra subject {ServiceAccount
management-infra management-admin }.
Info: clusterrolebinding/cluster-readers has extra subject {ServiceAccount
default router }.

Info: clusterrolebinding/self-provisioners has more subjects than expected.

 Use the `oadm policy reconcile-cluster-role-bindings` command to update
 the role binding to remove extra subjects.

Info: clusterrolebinding/self-provisioners has extra subject {ServiceAccount
management-infra management-admin }.
[Note] Summary of diagnostics execution (version v3.6.173.0.49):
[Note] Completed with no errors or warnings seen.

```

The output tells you that although you have more subjects than are required, you do not have any errors to investigate.

- From the **workstation** VM, run the **lab diagnostics cleanup** command. The script ensures that the OpenShift Container Platform cluster is configured appropriately for the next exercise:

```
[student@workstation ~]$ lab diagnostics cleanup
```

---

This concludes the guided exercise.

# Backing up and Restoring Key Data Stores

## Objective

After completing this section, students should be able to back up and restore key data stores in an OpenShift cluster.

## Backup and Restore

OpenShift Container Platform provides a method for backing up and restoring the cluster. Whether backing up the cluster state to separate storage or restoring the cluster by recreating the state, both are done at the cluster level. The cluster state consists of the Etcd data on each master, API objects, registry storage, and volume storage.

### Prerequisites

Red Hat recommends saving all the files used in the initial installation which may include the following:

- `/etc/yum.repos.d/*` if the disconnected installation method was used
- `~/.config/openshift/installer.cfg.yml` if the quick installation method was used
- All the Ansible Playbooks if the advanced installation method was used
- The Ansible inventory file if the advanced installation method was used

### Etcd Data Directory

The location of the **etcd** data directory may vary depending on the method used to deploy **etcd**.

- An all-in-one cluster deployment uses `/var/lib/origin/openshift.local.etcd`
- An **etcd** instance uses `/var/lib/etcd`.

### Master Backup

To back up the master nodes, save all the keys and certificates for each host.

```
[user@master ~]$ cd /etc/origin/master
[user@master master]$ tar cf /tmp/certs-and-keys-master1.tar *.key *.crt
```

## Etcd Backup

It is important to synchronize the **etcd** data. Stop **etcd** on all the master node hosts running in the cluster:

```
[user@master ~]$ systemctl stop etcd
```

Use **etcdctl backup** to create an **etcd** backup. For example, to create a backup of `/var/lib/etcd` to `/etc/lib/etcd.bak`, run the following command:

```
[root@master master]# etcdctl backup --data-dir /var/lib/etcd \
```

```
--backup-dir /var/lib/etcd.bak
```

### Registry Certificates Backup

The registry certificates must be saved from every master and node. For example, to save all the registry certificates in **/etc/docker/certs.d/**, run the following command:

```
[user@master ~]$ cd /etc/docker/certs.d/
[user@master certs.d]$ tar cf /tmp/docker-registry-certs-$(hostname).tar *
```

### Project Backup

Although there is preliminary support for *per-project backups* currently, future releases of OpenShift Container Platform feature specific support for per-project back up and restore. However, there is a procedure that can be used to back up projects. The **oc export** command is used to back up API objects at the project level. Run the command for each object to be saved. For example, to back up the front-end deployment configuration file called **frontend** as **dc-frontend** in YAML format, run the following command:

```
[user@master ~]$ oc export dc frontend -o yaml > dc-frontend.yaml
```

Create a backup of the entire project. To back up the entire project to a file called **project.yaml**, use the following command:

```
[user@master ~]$ oc export all -o yaml > project.yaml
```



### Note

The **oc export all** command backs up the entire project with the exception of cluster objects such as namespaces and projects.

### Role Bindings

During the course of an OpenShift Container Platform cluster deployment an administrator may have created custom policy role bindings to modify user access to the project. In these situations the role bindings are required to be backed up as well. When the project is restored, the custom role bindings are restored, ensuring that the newly restored project has the appropriate role bindings configured. For example, to save the custom role bindings to **rolebindings.yaml**, run the following command:

```
[user@master ~]$ oc get rolebindings -o yaml --export=true > rolebindings.yaml
```

The following is a list of additional objects in a project that must be saved with the **oc get** command.

- Role bindings
- Service accounts
- Persistent volume claims
- Secrets

### Application Data Backup

Use the **oc rsync** command to back up the application data when **rsync** is installed within a container. You can also use storage solutions such as Cinder, Gluster, and NFS.

### Cluster Restore

To restore the cluster for single member **etcd** clusters, perform the following procedure:

1. Reinstall OpenShift exactly as the previous install.
2. Complete all postinstallation steps.
3. Restore all the certificates and keys for each master node host:

```
[root@master ~]# cd /etc/origin/master
[root@master master]# tar xvf /tmp/certs-and-keys-master1.tar
```

4. Stop the **etcd** service to start the restore process.

```
[root@master master]# systemctl stop etcd
```

5. Restore **etcd** from the backup:

```
[root@master master]# mv /var/lib/etcd /var/lib/etcd.orig
[root@master master]# cp -Rp /var/lib/etcd.bak /var/lib/etcd
[root@master master]# chcon -R --reference /var/lib/etcd.orig /var/lib/etcd
[root@master master]# chown -R etcd:etcd /var/lib/etcd
```

6. Start the **etcd** service to restore the data store.

```
[root@master master]# systemctl start etcd
```

### Project Restore

To restore a project, you must recreate the entire project and all of the objects that were exported during the backup procedure. Use the **oc create** command to restore the objects that were saved for backup. For example, to restore the role bindings, use the following command:

```
[user@master ~]$ oc create -f rolebindings.yaml
```

This concludes the demonstration.

## References

Further information is available in the *Backup and Restore* section of the *OpenShift Container Platform Cluster Administration* guide for OpenShift Container Platform 3.6 available at  
<https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Backing Up and Restoring Key Data Stores

In this exercise, you will backup the cluster and a project.

## Outcomes

You should be able to:

- Backup the **cluster-backup** project.
- Backup deployment configurations, role bindings, accounts, secrets, and image streams.
- Restore a project.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab backup setup** command. The script ensures that the OpenShift Container Platform cluster is correctly configured for this exercise:

```
[student@workstation ~]$ lab backup setup
```

## Steps

1. On the **workstation** VM, log in to the cluster as the **developer** user with a password of **redhat**, and create a project called **cluster-backup**.
  - 1.1. On the **workstation** VM, run the **oc login** command to log in to the cluster:

```
[student@workstation ~]$ oc login \
https://console.lab.example.com \
-u developer -p redhat
Login successful.
```

```
You don't have any projects. You can try to create a new project, by running
oc new-project <projectname>
```

- 1.2. On the **workstation** VM, run the **oc new-project** command to create the project:

```
[student@workstation ~]$ oc new-project cluster-backup
Now using project "cluster-backup" on server "https://
master1.lab.example.com:443".
... output omitted ...
```

- 1.3. Create the **cluster-app** application.

Use the **hello-openshift** image as the base image and use the **--insecure-registry** option because offline registry is insecure:

```
[student@workstation ~]$ oc new-app --name=cluster-app \
--docker-image=registry.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry
```

```
--> Found Docker image ddaec72 (23 minutes old) from
 registry.lab.example.com:5000 for "registry.lab.example.com:5000/openshift/
 hello Openshift"
 ... output omitted ...

--> Creating resources ...
 imagestream "cluster-app" created
 deploymentconfig "cluster-app" created
 service "cluster-app" created
--> Success
 Run 'oc status' to view your app.
```



## Note

The file **/home/student/D0380/labs/backup/commands.txt** provides the command for copying and pasting.

## 2. Backup the cluster files.

### 2.1. Use the **ssh** command to log in to the **master1** VM as the **root** user:

```
[student@workstation ~]$ ssh root@master1
```

### 2.2. Create the **certs-and-keys-master1.tar** file and save all certificates and keys from the **/etc/origin/master** directory.

```
[root@master1 ~]# mkdir -p ~/D0380/cluster-backup
[root@master1 ~]# cd /etc/origin/master
[root@master1 master]# tar cf ~/D0380/cluster-backup/ \
certs-and-keys-master1.tar \
*.key *.crt
```

### 2.3. Save all the registry certificates.

```
[root@master1 master]# tar cf /root/D0380/cluster-backup/docker-registry-
certs.tar /etc/docker/certs.d/*
```

## 3. Backup the Etcd database.

### 3.1. Stop the **etcd** service on all the **master** VMs to ensure Etcd data is fully synchronized:

```
[root@master1 master]# for i in {1..3}; \
do ssh root@master${i} "systemctl stop etcd"; done
```



## Note

You may be prompted to accept the key and input the password. Accept the key and use **redhat** as the password.

- 
- 3.2. Go to the backups directory and create the **etcd** directory in **cluster-backup**. Use the **etcdctl backup** command to back up the **/var/lib/etcd** directory to **/etc/lib/etcd.bak**:

```
[root@master1 master]# cd /root/D0380/cluster-backup
[root@master1 cluster-backup]# mkdir etcd
[root@master1 cluster-backup]# etcdctl backup --data-dir /var/lib/etcd \
--backup-dir ~/D0380/cluster-backup/etcd/etcd.bak
```

- 3.3. Copy the **db** file to the **etcd** directory.

```
[root@master1 cluster-backup]# cp /var/lib/etcd/member/snap/db etcd/
```

- 3.4. Start the **etcd** service on all the **master** VMs:

```
[root@master1 cluster-backup]# for i in {1..3}; \
do ssh root@master${i} "systemctl start etcd"; done
```



### Note

The service can take some time to start, and might result in a timeout message. You can disregard this message.



### Note

You may be prompted to accept the key and input the password. Accept the key and use **redhat** as the password.

4. Backup the cluster objects to YAML files.

- 4.1. Use the **oc export** command to backup the **cluster-backup** project to **project\_backup.yaml**:

```
[root@master1 cluster-backup]# oc export project cluster-backup -o yaml \
> project_backup.yaml
```

- 4.2. Use the **oc get** command to backup the **cluster-app** deployment configuration to **dc\_user-backup.yaml**:

```
[root@master1 cluster-backup]# oc export dc cluster-app -n cluster-backup \
-o yaml > dc_user-backup.yaml
```

- 4.3. Export the role bindings to **roles\_backup.yaml**:

```
[root@master1 cluster-backup]# oc get rolebindings -n cluster-backup -o yaml \
--export=true > roles-backup.yaml
```

4.4. Export the service accounts to **accounts\_backup.yaml**:

```
[root@master1 cluster-backup]# oc get serviceaccount -n cluster-backup -o yaml \
--export=true > accounts-backup.yaml
```

4.5. Export the secrets to **secrets\_user-backup.yaml**:

```
[root@master1 cluster-backup]# oc get secret -n cluster-backup -o yaml \
--export=true > secrets_user-backup.yaml
```

4.6. Export the service to **service-user-backup.yaml**:

```
[root@master1 cluster-backup]# oc get service -n cluster-backup -o yaml \
--export=true > service_user-backup.yaml
```

4.7. Export the image stream to **is-user-backup.yaml**:

```
[root@master1 cluster-backup]# oc get is -n cluster-backup -o yaml \
--export=true > is_user-backup.yaml
```

## 4.8. List the files in the directory. Ensure that the following files are present:

```
[root@master1 cluster-backup]# ls -al
total 172
drwxr-xr-x. 3 root root 4096 Feb 7 12:41 .
drwxr-xr-x. 3 root root 28 Feb 7 09:53 ..
-rw-r--r--. 1 root root 1344 Feb 7 12:40 accounts-backup.yaml
-rw-r--r--. 1 root root 71680 Feb 7 12:05 certs-and-keys-master1.tar
-rw-r--r--. 1 root root 1748 Feb 7 12:40 dc_user-backup.yaml
-rw-r--r--. 1 root root 10240 Feb 7 12:06 docker-registry-certs.tar
drwxr-xr-x. 3 root root 32 Feb 7 12:07 etcd
-rw-r--r--. 1 root root 1404 Feb 7 12:41 is_user-backup.yaml
-rw-r--r--. 1 root root 444 Feb 7 12:39 project_backup.yaml
-rw-r--r--. 1 root root 2027 Feb 7 12:40 roles-backup.yaml
-rw-r--r--. 1 root root 60852 Feb 7 12:40 secrets_user-backup.yaml
-rw-r--r--. 1 root root 825 Feb 7 12:40 service_user-backup.yaml
```

4.9. Exit from **master1**.

```
[root@master1 cluster-backup]#exit
```

## 5. Restore the cluster resources.

5.1. From **workstation** log in to the cluster as the **admin** user and delete the **cluster-backup** project.

```
[student@workstation ~]$ oc login \
https://console.lab.example.com \
-u developer -p redhat
Login successful.

You have one project on this server: "cluster-backup"
```

```
Using project "cluster-backup".
[student@workstation ~]$ oc delete project cluster-backup
project "cluster-backup" deleted
```

- 5.2. From **workstation**, as the **student** user, use the **scp** command to retrieve the **cluster-backup** directory to **D0380**.

```
[student@workstation ~]$ scp -r root@master1:D0380/cluster-backup D0380/
... output omitted ...
```

- 5.3. To restore a project, start by recreating the project as the **developer** user.

```
[student@workstation ~]$ oc new-project cluster-backup
```



### Note

If the following message appears, wait a couple of minutes before retrying.

```
Error from server (AlreadyExists): project.project.openshift.io
"cluster-backup" already exists
```

- 5.4. From workstation, go to the **/home/student/D0380/cluster-backup** directory and edit the **service\_user-backup.yaml** file. In the **spec** group, remove the IP address defined for the **clusterIP** value and save the file.

```
spec:
 clusterIP:
```

- 5.5. Define a **for** loop to recreate the resources:

```
[student@workstation cluster-backup]$ for file in *user-backup.yaml; do oc
 create -f $file; done
deploymentconfig "cluster-app" created
imagestream "cluster-app" created
secret "builder-dockercfg-h6q8j" created
secret "builder-token-3kdb6" created
secret "builder-token-7hkr9" created
secret "default-dockercfg-bvzd7" created
secret "default-token-f30vg" created
secret "default-token-x3d19" created
secret "deployer-dockercfg-sht8n" created
secret "deployer-token-k63fj" created
secret "deployer-token-ph6p5" created
service "cluster-app" created
```

6. Ensure that the application is reachable.

- 6.1. Ensure that there is a pod running in the project.

```
[student@workstation cluster-backup]$ oc get pods
```

| NAME                | READY | STATUS  | RESTARTS | AGE |
|---------------------|-------|---------|----------|-----|
| cluster-app-1-954h3 | 1/1   | Running | 0        | 23s |

6.2. Expose the application.

```
[student@workstation cluster-backup]$ oc expose svc cluster-app \
--hostname=cloud.apps.lab.example.com
route "cluster-app" exposed
```

6.3. Use the **curl** command to retrieve the content from the application.

```
[student@workstation cluster-backup]$ curl cloud.apps.lab.example.com
```

7. Clean up the lab.

Delete the **cluster-backup** and **new-cluster-install** projects:

```
[student@workstation cluster-backup]$ oc delete project cluster-backup
```

8. This concludes the guided exercise.

# Cleaning up the Cluster

## Objective

After completing this section, students should be able to describe and perform clean-up activities on a cluster.

## Garbage Collection

OpenShift Container Platform uses a mechanism called *garbage collection* to clean up unused images and containers.

Garbage collection is categorized into two types:

1. *Container garbage collection*: Runs every minute, and eliminates terminated containers.
2. *Image garbage collection*: Runs every five minutes and removes images not referenced by any running pods.

### Container Garbage Collection

Container garbage collection uses settings in the policy that specify at which point in their life cycle containers are removed. The **kubeletArguments** section of the **/etc/origin/node/node-config.yaml** file on each node configures these settings. There are three settings within the policy that are used for container garbage collection:

#### **maximum-dead-containers**

The maximum number of total dead containers in the node. The default is 240.

#### **minimum-container-ttl-duration**

The minimum age that a container is eligible for garbage collection. The default is one minute. Use zero (0) for no limit. Values for this setting can be specified using unit suffixes such as h for hours, m for minutes, and s for seconds.

#### **maximum-dead-containers-per-container**

The number of instances to retain per pod container. The default is two.

The **maximum-dead-containers** setting takes precedence over the **maximum-dead-containers-per-container** setting. When there is a conflict in the settings, a special consideration is given. During the process of removing a dead container from a node, the container's files are removed and only containers created by that specific node are garbage collected.

## Detecting Containers for Deletion

Each execution of the garbage collection loop uses the following steps:

1. Retrieve a list of available containers.
2. Filter out all containers that are running or are not alive longer than the **minimum-container-ttl-duration** parameter.
3. Classify all remaining containers into equivalent classes based on pod and image name membership.

4. Remove all unidentified containers (containers that are managed by Kubelet but their name is malformed).
5. For each class that contains more containers than the **maximum-dead-containers-per-container** parameter, sort containers in the class by creation time.
6. Start removing containers from the oldest first until the **maximum-dead-containers-per-container** parameter is met.
7. If there are still more containers in the list than the **maximum-dead-containers** parameter, the collector starts removing containers from each class so the number of containers in each one is not greater than the average number of containers per class, or **<all\_remaining\_containers>/<number\_of\_classes>**.

#### Image Garbage Collection

Image garbage collection receives reports of disk usage on a node from *cAdvisor*, a performance analysis agent that is integrated with OpenShift Container Platform. Image garbage collection uses cAdvisor data to decide whether an image should be removed. The **kubeletArguments** section in the `/etc/origin/node/node-config.yaml` file defines the image garbage collection settings. There are two settings that configure image garbage collection:

##### **image-gc-high-threshold**

The percentage of disk usage, expressed as an integer, that triggers image garbage collection. The default is 90.

##### **image-gc-low-threshold**

The percentage of disk usage, expressed as an integer, that image garbage collection attempts to free. Default is 80.

## Detecting Images for Deletion

There are two lists of images retrieved when image garbage collection is executed. One list consists of images currently running in at least one pod while the other lists images available on a host. The following steps are taken to detect images for deletion:

1. All images are marked with a time stamp.
2. If the image is running (the first list) or is newly detected (the second list), it is marked with the current time.
3. The remaining images are marked from the previous executions.
4. All images are then sorted by the time stamp.
5. When garbage collection begins, the oldest images are deleted first. Garbage collection continues until the stopping criterion is met.

## Pruning

Through normal user operations, API objects created in OpenShift Container Platform can accumulate in the Etcd data store.

OpenShift Container Platform has a process for deleting older images and layers that are no longer in use but are still using disk space. This process is known as pruning.

Pruning is performed on three object types: images, builds, and deployments.

### Pruning Deployments

To prune deployments config objects, use the following command:

```
[user@demo ~]$ oc adm prune deployments
```

#### Prune Deployment Options

The following table lists the options and their descriptions for pruning deployments.

| Option                         | Description                                                                                                                     |
|--------------------------------|---------------------------------------------------------------------------------------------------------------------------------|
| --confirm                      | Indicate that pruning should occur, instead of performing a dry run.                                                            |
| --orphans                      | Prune all deployments whose deployment configuration no longer exists, status is complete or failed, and replica count is zero. |
| --keep-complete=<N>            | Per deployment configuration, keep the last N deployments whose status is complete and replica count is zero (default is five). |
| --keep-failed=<N>              | Per deployment config, keep the last N deployments whose status is failed and replica count is zero (default is one).           |
| --keep-younger-than=<duration> | Do not prune any object that is younger than <duration> relative to the current time (default is 60m)                           |

To identify which deployment configurations are deleted by the pruning operation, run the following command:

```
[user@demo ~]$ oc adm prune deployments \
--orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m
```

To perform the pruning operation, run the following command:

```
[user@demo ~]$ oc adm prune deployments \
--orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m --confirm
```

### Pruning Builds

Builds that are no longer required by OpenShift Container Platform due to age and status are pruned. Pruning builds uses the same options as pruning deployments. To prune builds, run the following command:

```
[user@demo ~]$ oc adm prune builds
```

To list what a pruning operation would delete, run the following command:

```
[user@demo ~]$ oc adm prune builds \
--orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m
```

To perform the pruning operation on builds, run the following command:

```
[user@demo ~]$ oc adm prune builds \
```

```
--orphans --keep-complete=5 --keep-failed=1 \
--keep-younger-than=60m --confirm
```

### Pruning Images

To prune images that are no longer required by the system due to age and status, run the following command:

```
[user@demo ~]$ oc adm prune images
```

For example, to perform a dry run of pruning images, keeping up to five revision tags, and keeping resources such as images, image streams, and pods younger than 120 minutes, run the following command:

```
[user@demo ~]$ oc adm prune images \
--keep-tag-revisions=5 --keep-younger-than=120m
```

To perform the pruning action, run the following command:

```
[user@demo ~]$ oc adm prune images \
--keep-tag-revisions=5 --keep-younger-than=120m --confirm
```



### Note

Currently, to prune images you must first log in to the CLI as a user with an access token. You must also have the cluster **rolesystem:image-pruner** role or with other role with higher privileges (for example, **cluster-admin**).

## Image Prune Conditions

OpenShift Container Platform manages images with the annotation **openshift.io/image.managed**. Image pruning removes any image that was created at least **--keep-younger-than** minutes ago and that exceeds the smallest limit defined in the same project that is not currently referenced by any image pruning conditions. References to the image are removed from all image streams that have a reference to the image in **status.tags**.

### Viewing Image Statistics

Image statistics for images that are managed by OpenShift Container Platform are available using the **oc adm top** command. This includes usage statistics for all images to the internal registry. To view the usage statistics for the image stream:

```
[user@demo ~]$ oc adm top images
NAME IMAGESTREAMTAG USAGE ...
sha256:4ca0781f9888f10e2967 openshift/python (3.5) 211.60MiB ...
sha256:ed314335be90b20296ce ... 1.97MiB ...
sha256:e733c82e5a87ce68f33f ... 205.67MiB ...
```

### Viewing Image Stream Statistics

OpenShift Container Platform manages image stream statistics for images and they are available using the **oc adm top imagestreams** command. For example, to view the usage image stream statistics, run the following command:

```
[user@demo ~]$ oc adm top imagestreams
NAME STORAGE IMAGES LAYERS
openshift/python 2.12GiB 5 36
openshift/ruby 829.56MiB 6 27
```



## References

Further information is available in the *Garbage Collection* section of the *Cluster Administration* guide for OpenShift Container Platform 3.6 available at  
| <https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Cleaning up the Cluster

In this exercise, you will cleanup the cluster by pruning images, builds, and deployments.

## Outcomes

You should be able to remove all unnecessary items stored in the OpenShift Container Platform environment to save disk space.

### Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab prune setup** command. The script ensures that the OpenShift Container Platform cluster is correctly configured for this exercise:

```
[student@workstation ~]$ lab prune setup
```

### Steps

1. Ensure that the internal image registry from the **services.lab.example.com** host contains the **php-70** image.

On the **workstation** VM, use the **docker-registry-cli** command to browse the internal image registry at **registry.lab.example.com** on port 5000. Use the **search** option to browse images in the registry:

```
[student@workstation ~]$ docker-registry-cli \
registry.lab.example.com:5000 search php-70

1) Name: rhscl/php-70-rhel7
Tags: latest

1 images found !
```

2. Create an application using the **php70** image.

- 2.1. Create a project called **cluster-prune**.

On the **workstation** VM, use the **oc login** command to log in to the cluster as the **developer** user with a password of **redhat**.

```
[student@workstation cluster-review]$ oc login \
https://console.lab.example.com \
-u developer -p redhat
Login successful.

You don't have any projects. You can try to create a new project, by running
oc new-project <projectname>
```

- 2.2. Create the **cluster-prune** project.

On the **workstation** VM, use the **oc new-project** command to create the project:

```
[student@workstation ~]$ oc new-project cluster-prune
```

```
Now using project "cluster-prune" on server "https://
console.lab.example.com:443".
...
```

### 2.3. Create the **prune-app** application.

The **prune-app** application is a PHP application that is built by the OpenShift Source-to-Image (S2I) process. It uses the **php70** image as the base image. Use the **--insecure-registry** option, because the offline registry does not use a secure communication channel:

```
[student@workstation ~]$ oc new-app \
http://services.lab.example.com/php-helloworld -i php:7.0 \
--name=prune-app
--> Found image c8160a6 (3 weeks old) in image stream "openshift/php" under tag
"7.0" for "php:7.0"
...output omitted...

--> Creating resources ...
imagestream "prune-app" created
deploymentconfig "prune-app" created
service "prune-app" created
--> Success
Run 'oc status' to view your app.
```



#### Note

The command may be copied and pasted from **/home/student/D0380/labs/prune/commands.txt**.

### 2.4. Use the **oc logs -f bc/prune-app** command to track the application creation progress:

```
[student@workstation ~]$ oc logs -f bc/prune-app
Cloning "http://services.lab.example.com/php-helloworld" ...
Commit: 30cf097b4e68f9ec260055f56a7dfbeb1a56c47c (Establish remote repository)
Author: root <root@services.lab.example.com>
Date: Thu Nov 9 18:35:25 2017 -0500
---> Installing application source...
Pushing image docker-registry.default.svc:5000/prune-final/prune-app:latest ...
Pushed 0/6 layers, 6% complete
Pushed 1/6 layers, 32% complete
Pushed 2/6 layers, 52% complete
Pushed 3/6 layers, 64% complete
Pushed 4/6 layers, 82% complete
Pushed 5/6 layers, 100% complete
Pushed 6/6 layers, 100% complete
Push successful
```

3. Remove the **prune-app** application images from the OpenShift Container Platform runtime environment.

- 3.1. Use the **oc delete project** command to delete the **cluster-prune** project:

```
[student@workstation ~]$ oc delete project cluster-prune
project "cluster-prune" deleted
```

- 3.2. Log in as the **admin** user with the **redhat** password to prune images.

```
[student@workstation cluster-review]$ oc login \
https://console.lab.example.com \
-u admin -p redhat
```

- 3.3. Use the **oc adm prune images** command to evaluate which images are going to be deleted.

```
[student@workstation ~]$ oc adm prune images \
--registry-url=http://registry.lab.example.com:5000 --keep-younger-than=30s
Dry run enabled - no modifications will be made. Add --confirm to remove images

Deleting registry layer blobs ...
BLOB
sha256:6d7ccc0c36f2ba8c0b7dce6d9ca0ad83d6ccc00c407330fb18452fc45d96083e
sha256:64cecf0f039494607ac6556361effdcfdcae3e7c47354c9737ad97a352e18c55e
sha256:e7e4a9d05f7e77fd2ab60d04afcc66b9bc320ec9a40849274934ca5c0fa12174
sha256:7602ef516f2c199f37041903697b7a192ef9e6cb1b2e2f84d3db22dbda0c4f67

Deleting images from server ...
IMAGE
sha256:8c94d5da00a8933773c6e3731715249db4feef906479fc4db7108f6f48ef2110
sha256:994e1bf35665cbaed21cea3652fc7a35b1ebbd343a758260c11d344172fbad70
```



## Note

The command may not return any image. It may take some time for them to show.

- 3.4. Use the **oc adm prune images** command with the **--confirm** option and the **--registry-url** to actually prune the **prune-apps** image.

```
[student@workstation ~]$ oc adm prune images \
--confirm --registry-url=http://registry.lab.example.com:5000 \
--keep-younger-than=30s

Deleting registry layer blobs ...
BLOB
sha256:c12ba2faea4a9e5fffbf58248e060abe8871707c750b3933036f41ce61d53acf
sha256:84ab71c018f9110f6694462367272816e9c4c7b6c417ecef496f475e629f5c64

Deleting images from server ...
IMAGE
sha256:fd9b8aba67db4d7948d4050c32c8bc4d8879291b38190e0e79c0c1cd6126c3ae
```

This concludes this exercise.

# Adding and Removing Cluster Nodes

## Objective

After completing this section, students should be able to add and remove nodes from the cluster.

## Adding Hosts

OpenShift Container Platform with Ansible provides the `/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-master/scaleup.yml` Ansible Playbook that is used to add nodes to a cluster. The `scaleup.yml` Ansible Playbook adds nodes to the cluster by:

1. Querying the master nodes
2. Generating new certificates for the hosts
3. Distributing the new certificates for the hosts
4. Running the configuration playbooks on only the new hosts

## Adding Hosts

To add new hosts to an existing cluster, perform the following steps:

1. Update the `atomic-openshift-utils` package:

```
[user@demo]$ yum update atomic-openshift-utils
```

2. In the inventory file, define the `new_<host_type>` group in the `[OSEv3:children]` section. For example, to add a new node host, define the `new_nodes` group.



### Note

To add new master hosts, define the `new_masters` group in the `[OSEv3:children]` section.

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
new_nodes
```

3. Create a section called `new_<host_type>`. The new section contains information for any new hosts added to the existing cluster. For example, to add the new node hosts `node1.demo.example.com` to an existing cluster, add it to the `new_nodes` section in the inventory file:

```
...output omitted...
[new_nodes]
```

```
node1.demo.example.com openshift_node_labels="{'region': 'primary', 'zone': 'local'}"
openshift_ip=172.16.100.32 openshift_public_ip=10.25.1.32
...output omitted...
```

To add Etcd hosts, define the **new\_etcd** group in the **[OSEv3:children]** section:

```
[OSEv3:children]
masters
etcd
nodes
nfs
lb
new_etcd
```

4. Create a section called **new\_<host\_type>**. The new section contains information for any new hosts added to the existing cluster. For example, to add the new Etcd host **etcd1.demo.example.com** to an existing cluster, add it to the **new\_etcd** section in the inventory file:

```
...output omitted...
[new_etcd]
etcd1.demo.example.com openshift_node_labels="{'region': 'primary', 'zone': 'local'}"
openshift_ip=172.16.100.12 openshift_public_ip=10.25.1.42
...output omitted...
```

Adding new master hosts requires that the hosts added to the **new\_masters** section are added to the **new\_nodes** section as well. For example, to add the new master host **master1.demo.example.com** to an existing cluster, add **master3.demo.example.com** to both the **new\_masters** and **new\_nodes** section:

```
...output omitted...
[masters]
master1.demo.example.com openshift_ip=172.16.100.23 openshift_public_ip=10.25.1.33
master2.demo.example.com openshift_ip=172.16.100.24 openshift_public_ip=10.25.1.34

[new_masters]
master3.demo.example.com openshift_ip=172.16.100.25 openshift_public_ip=10.25.1.35

[new_nodes]
master3.demo.example.com openshift_ip=172.16.100.25 openshift_public_ip=10.25.1.35
...output omitted...
```

5. To add additional node hosts, run the **/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-node/scaleup.yml** Ansible Playbook:

```
[user@demo]$ ansible-playbook -i inventory_file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-node/scaleup.yml
```

To add additional master hosts, run the **/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-master/scaleup.yml** Ansible Playbook:

```
[user@demo]$ ansible-playbook -i inventory_file \
```

```
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-master/scaleup.yml
```

To add additional etcd hosts, run the **/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-etcd/scaleup.yml** Ansible Playbook:

```
[user@demo]$ ansible-playbook -i inventory_file \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-etcd/scaleup.yml
```

### Deleting Nodes from an Existing Cluster

Use the **oc delete node** command to delete nodes. The command deletes the node object in Kubernetes, however the pods that exist on the node itself remain.

To remove a node from a cluster, perform the following procedure:

1. Use the **oc adm manage-node** command to mark the node as unscheduled:

```
[user@demo]$ oc adm manage-node --schedulable=false node1.demo.example.com
```

Use the **oc adm drain** command to prepare the node for maintenance. For example, prior to removing **node1.demo.example.com** node, run the following command:

```
[user@demo]$ oc adm drain node1.demo.example.com
node "node1.demo.example.com" cordoned
node "node1.demo.example.com" drained
```

2. Use the **oc delete node** command to remove a node host from an existing cluster. For example, to remove the **node1.demo.example.com** node, run the following command:

```
[user@demo]$ oc delete node node1.demo.example.com
node "node1.demo.example.com" deleted
```

3. Update the inventory file to remove the node reference to the deleted node.

Delete in all the node groups references to the node that was removed.



## References

Further information is available in the *Adding Hosts* section of the *OpenShift Container Platform Cluster Administration* guide for OpenShift Container Platform 3.6 available at

  | <https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the *Removing Hosts* section of the *OpenShift Container Platform Cluster Administration* guide for OpenShift Container Platform 3.6 available at

  | <https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Adding and Removing Cluster Nodes

In this exercise, you will add and remove cluster nodes.

## Outcomes

You should be able to remove a node from the cluster and replace it using the **scaleup.yml** playbook.

### Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab add-node setup** command. The script ensures that the OpenShift Container Platform cluster is correctly configured for this exercise:

```
[student@workstation ~]$ lab add-node setup
```

## Steps

1. Log in to the cluster as **admin** using **redhat** as the password:

- 1.1. On the **workstation** VM, run the **oc login** command to log in to the cluster:

```
[student@workstation ~]$ oc login \
https://console.lab.example.com \
-u admin -p redhat
You have access to the following projects and can switch between them with 'oc
project <projectname>':
* default
glusterfs
kube-public
kube-system
logging
management-infra
openshift
openshift-infra

Using project "default".
```

2. View the current running pods.

- 2.1. Use the **oc get pods** command to view the running pods:

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
docker-registry-1-1jjlv 1/1 Running 2 5d
docker-registry-1-8rsnk 1/1 Running 2 5d
docker-registry-1-q80m4 1/1 Running 2 5d
registry-console-1-0jt4c 1/1 Running 2 5d
router-3-7dmf0 1/1 Running 0 16m
router-3-mn86w 1/1 Running 0 14m
router-3-q197g 1/1 Running 0 15m
```

---

3. View the current running nodes.

3.1. Use the **oc get nodes** command to view the running nodes:

```
[student@workstation ~]$ oc get nodes
NAME STATUS AGE VERSION
apps1.lab.example.com Ready 5d v1.6.1+5115d708d7
apps2.lab.example.com Ready 41m v1.6.1+5115d708d7
infra1.lab.example.com Ready 5d v1.6.1+5115d708d7
infra2.lab.example.com Ready 5d v1.6.1+5115d708d7
infra3.lab.example.com Ready 5d v1.6.1+5115d708d7
master1.lab.example.com Ready, SchedulingDisabled 5d v1.6.1+5115d708d7
master2.lab.example.com Ready, SchedulingDisabled 5d v1.6.1+5115d708d7
master3.lab.example.com Ready, SchedulingDisabled 5d v1.6.1+5115d708d7
```

4. Remove the **apps2** node from the cluster.

4.1. Use the **oc adm manage-node** command to unschedule the **apps2** node for removal:

```
[student@workstation ~]$ oc adm manage-node --schedulable=false \
apps2.lab.example.com
NAME STATUS AGE VERSION
apps2.lab.example.com Ready, SchedulingDisabled 13h v1.6.1+5115d708d7
```

4.2. Use the **oc adm drain** command to prepare **apps2** for removal:

```
[student@workstation ~]$ oc adm drain apps2.lab.example.com --ignore-daemonsets
node "apps2.lab.example.com" cordoned
WARNING: Ignoring DaemonSet-managed pods: logging-fluentd-1vjkd
node "apps2.lab.example.com" drained
```



### Note

The Fluentd is part of the logging installation. To remove it, use the **--ignore-daemonset** option.

4.3. Remove the **apps2** node from the cluster:

```
[student@workstation ~]$ oc delete node apps2.lab.example.com
node "apps2.lab.example.com" deleted
```

4.4. Navigate to the **Online Lab** tab in the ROLE Web UI. From the **ACTION** list for the **apps2** VM, select **Reset**. Click **YES, RESET IT** to confirm.

4.5. Use the **oc get nodes** command to verify the removal of the **apps2** node:

```
[student@workstation ~]$ oc get nodes
NAME STATUS AGE VERSION
apps1.lab.example.com Ready 5d v1.6.1+5115d708d7
infra1.lab.example.com Ready 5d v1.6.1+5115d708d7
infra2.lab.example.com Ready 5d v1.6.1+5115d708d7
infra3.lab.example.com Ready 5d v1.6.1+5115d708d7
```

|                         |                           |    |                   |
|-------------------------|---------------------------|----|-------------------|
| master1.lab.example.com | Ready, SchedulingDisabled | 5d | v1.6.1+5115d708d7 |
| master2.lab.example.com | Ready, SchedulingDisabled | 5d | v1.6.1+5115d708d7 |
| master3.lab.example.com | Ready, SchedulingDisabled | 5d | v1.6.1+5115d708d7 |

5. Use the **scaleup.yml** Ansible Playbook to add the **apps2** node into the cluster.

- 5.1. Use ssh to access the **console** VM as root using **redhat** as the password.

```
[student@workstation ~]$ ssh root@console
[root@console ~]# cd openshift-install/lab.example.com
```

- 5.2. Create the **new\_nodes** group in the **/root/openshift-install/lab.example.com/hosts-ucf** inventory file on the **console** node:

```
[OSEv3:children]
...
lb
glusterfs
glusterfs_registry
new_nodes
```

- 5.3. Create the **new\_nodes** section and move **apps2** from the **[nodes]** section to the **new\_nodes** section in the **/root/openshift-install/lab.example.com/hosts-ucf** inventory file on. Save your changes and exit the editor.

```
...
[new_nodes]
apps2.lab.example.com openshift_node_labels="{'region': 'primary', 'zone':
'local'}" openshift_ip=172.25.250.32 openshift_public_ip=172.25.250.32
...
... output omitted ...
```

- 5.4. From **workstation**, open a new terminal and run the **lab add-node grade** command as the **student** user to ensure that the inventory file contains the expected values. If the command returns an error, make the necessary changes to the inventory file.

```
[student@workstation ~]$ lab add-node grade
...
Comparing Entries in [OSEv3:vars]

 • Checking openshift_deployment_type..... PASS
 • Checking deployment_type..... PASS
 • Checking openshift_release..... PASS
 • Checking osm_cluster_network_cidr..... PASS
 • Checking openshift_portal_net..... PASS
 ...
...
```

- 5.5. From the **console** VM, run the **scaleup.yml** Ansible Playbook to add the **apps2** node to the cluster:

```
[root@console lab.example.com]# ansible-playbook -i hosts-ucf \
/usr/share/ansible/openshift-ansible/playbooks/byo/openshift-node/scaleup.yml
PLAY RECAP ****
apps2.lab.example.com : ok=205 changed=15 unreachable=0 failed=0
```

---

|                         |   |       |           |               |          |
|-------------------------|---|-------|-----------|---------------|----------|
| localhost               | : | ok=14 | changed=0 | unreachable=0 | failed=0 |
| master1.lab.example.com | : | ok=32 | changed=1 | unreachable=0 | failed=0 |
| master2.lab.example.com | : | ok=13 | changed=1 | unreachable=0 | failed=0 |
| master3.lab.example.com | : | ok=13 | changed=1 | unreachable=0 | failed=0 |

5.6. Revert the changes from the Ansible inventory file.

Run the **git checkout** command to revert the changes:

```
[root@console lab.example.com]# git checkout hosts-ucf
```

5.7. Exit from the **console** VM:

```
[root@console lab.example.com]# exit
[student@workstation ~]$
```

5.8. Use the **oc get nodes** command to confirm the **apps2** node is a cluster member:

```
[student@workstation ~]$ oc get nodes
NAME STATUS AGE VERSION
apps1.lab.example.com Ready 5d v1.6.1+5115d708d7
apps2.lab.example.com Ready 41m v1.6.1+5115d708d7
infra1.lab.example.com Ready 5d v1.6.1+5115d708d7
infra2.lab.example.com Ready 5d v1.6.1+5115d708d7
infra3.lab.example.com Ready 5d v1.6.1+5115d708d7
master1.lab.example.com Ready,SchedulingDisabled 5d v1.6.1+5115d708d7
master2.lab.example.com Ready,SchedulingDisabled 5d v1.6.1+5115d708d7
master3.lab.example.com Ready,SchedulingDisabled 5d v1.6.1+5115d708d7
```

This concludes the guided exercise.

## Quiz: Maintaining An OpenShift Cluster

Match the items below to their counterparts in the table.

Backs up the **etcd** data directory.

Backups the front-end deployment configuration file front end as dc-frontend in YAML format.

Checks cluster roles and cluster role bindings for consistency with the base policy.

Drains nodes in preparation for maintenance.

Identifies the default master configuration file.

Provides image stream statistics.

Prunes deployment configuration objects.

| Command                                                    | Description |
|------------------------------------------------------------|-------------|
| <b>oc adm diagnostics ClusterRoleBindings</b>              |             |
| <b>/etc/origin/master/master-config.yaml</b>               |             |
| <b>oc export dc frontend -o yaml &gt; dc-frontend.yaml</b> |             |

---

| Command                         | Description |
|---------------------------------|-------------|
| <b>oc adm prune deployments</b> |             |
| <b>oc adm drain</b>             |             |
| <b>oc adm top imagestreams</b>  |             |
| <b>etcdctl backup</b>           |             |

## Solution

Match the items below to their counterparts in the table.

| Command                                                    | Description                                                                                  |
|------------------------------------------------------------|----------------------------------------------------------------------------------------------|
| <b>oc adm diagnostics ClusterRoleBindings</b>              | Checks cluster roles and cluster role bindings for consistency with the base policy.         |
| <b>/etc/origin/master/master-config.yaml</b>               | Identifies the default master configuration file.                                            |
| <b>oc export dc frontend -o yaml &gt; dc-frontend.yaml</b> | Backups the front-end deployment configuration file front end as dc-frontend in YAML format. |
| <b>oc adm prune deployments</b>                            | Prunes deployment configuration objects.                                                     |
| <b>oc adm drain</b>                                        | Drains nodes in preparation for maintenance.                                                 |
| <b>oc adm top imagestreams</b>                             | Provides image stream statistics.                                                            |
| <b>etcdctl backup</b>                                      | Backs up the <b>etcd</b> data directory.                                                     |

# Summary

In this chapter, you learned:

- A future releases of OpenShift Container Platform is likely feature specific support for per-project backup and restore.
- Application builds are pruned that are no longer required by OpenShift Container Platform due to age and status.
- Image pruning removes any image that was created at least **--keep-younger-than** minutes ago.
- The **oc delete node** command deletes the node object in Kubernetes, however the pods that exist on the node itself remain.
- The **oc adm diagnostics** command checks for the standard configuration files for client, master, and node, and if found, uses them for diagnostics.





## CHAPTER 8

# MANAGING SYSTEM RESOURCES

| Overview          |                                                                                                                                                                                                                                                                                       |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Goal</b>       | Manage operating system and cluster resources for optimal performance.                                                                                                                                                                                                                |
| <b>Objectives</b> | <ul style="list-style-type: none"><li>Configure various admission controllers to manage OpenShift resources.</li><li>Configure OpenShift to use the overcommitment and idling features.</li><li>Manage the consumption of operating system and OpenShift cluster resources.</li></ul> |
| <b>Sections</b>   | <ul style="list-style-type: none"><li>Configuring Admission Controllers (and Guided Exercise)</li><li>Configuring OpenShift for Overcommitment and Idling Services (and Guided Exercise)</li><li>Managing Operating System and OpenShift Resources (and Guided Exercise)</li></ul>    |
| <b>Lab</b>        | Managing System Resources                                                                                                                                                                                                                                                             |

# Configuring Admission Controllers

## Objective

After completing this section, students should be able to configure various admission controllers to manage OpenShift Container Platform resources.

## Defining Admission Controller Plug-ins

OpenShift Container Platform provides admission controller plug-ins as a way to manipulate requests sent to the master API and to customize the runtime environment. They intercept authenticated and authorized requests sent to the master API in OpenShift.

Some use cases are described below:

- *Update the API request:* Customize the values in the request, such as the object type names, to guarantee standardized OpenShift service names.
- *Update the runtime environment:* Change the quotas for CPU and disk usage for each project.
- *Update the environment:* Update OpenShift Container Platform configuration, such as changing the labels of all the applications.

Each admission controller plug-in is executed in sequence before a request is accepted into the cluster. If any admission controller in the sequence rejects the request, the entire request is rejected immediately, and an error is returned to the end user.

The OpenShift Container Platform master has a single, ordered admission controller chain for Kubernetes and OpenShift Container Platform resources, and it is configured in the **/etc/origin/master/master-config.yaml** file. To enable any change, the **atomic-openshift-master** service must be restarted on each master node.



### Important

Modifying the admission controller chain is not recommended unless you know exactly what you are doing. Future versions of the product may use a different set of plug-ins and may change their ordering.

Each admission controller is configured with the following configuration excerpt:

```
admissionConfig:
 pluginConfig:
 AlwaysPullImages: ①
 configuration:
 kind: DefaultAdmissionConfig
 apiVersion: v1
 disable: false ②
```

① Admission controller plug-in name.

② Indicates whether the admission controller is enabled or not.

# Customizing Provided Admission Controller Plug-ins

OpenShift Container Platform has the following admission controller plug-ins:

## **ProjectRequestLimit**

Limits the number of self-provisioned projects per user. The following excerpt configures two projects for any user and ten projects for users at the **premium** level:

```
admissionConfig:
 pluginConfig:
 ProjectRequestLimit:
 configuration:
 apiVersion: v1
 kind: ProjectRequestLimitConfig
 limits:
 - selector:
 level: premium
 maxProjects: 10
 - maxProjects: 2
```

As a use case for the previous configuration, you may label all paid customers as **level: premium**. This label identifies all users who may create up to ten projects. Any other customer, such as customers with a trial account, can only create two projects.

## **BuildOverrides**

Overrides the settings that were set for a specific build. As a security measure, the plug-in only allows image updates for users who have permission to pull them:

```
admissionConfig:
 pluginConfig:
 BuildOverrides:
 configuration:
 apiVersion: v1
 kind: BuildOverridesConfig
 forcePull: true
 nodeSelector:
 region: apac
```

This configuration forces everyone to deploy pods to nodes labeled **region: apac** due to a network outage in all other nodes.



## Note

You can also use the advanced installation Ansible inventory file to configure the admission controller plug-in.

## **BuildDefaults**

Configures the global defaults for settings and environment variables, such as the proxy configuration for Git cloning. The **BuildDefaults** admission controller overrides any values that were set for a specific build:

```
admissionConfig:
 pluginConfig:
 BuildDefaults:
```

```

configuration:
 apiVersion: v1
 kind: BuildDefaultsConfig
 resources:
 requests:
 cpu: "100m"
 memory: "256Mi"
 limits:
 cpu: "100m"
 memory: "256Mi"

```

This configuration defines resource limits, such as CPU and memory, for all pods that do not have this configuration.



### Note

You can also use the advanced installation Ansible inventory file to configure the admission controller plug-in.

#### **PodNodeConstraints**

Constrains the pod scheduling to a node selector. Change the **admissionConfig** and **kubernetesMasterConfig** sections in the **master-config.yaml** file to ensure the correct functioning of this admission controller:

```

admissionConfig:
 pluginConfig:
 PodNodeConstraints:
 configuration:
 apiVersion: v1
 kind: PodNodeConstraintsConfig
 nodeSelectorLabelBlacklist:
 - kubernetes.io/hostname
 - region: us-west

...output omitted...

kubernetesMasterConfig:
 admissionConfig:
 pluginConfig:
 PodNodeConstraints:
 configuration:
 apiVersion: v1
 kind: PodNodeConstraintsConfig
 nodeSelectorLabelBlacklist:
 - kubernetes.io/hostname
 - region: us-west

```

This configuration blocks all nodes with the **region: us-west** label from pod scheduling. This is another way of preventing pod scheduling in nodes in a specific region.

#### **PodNodeSelector**

Schedules some pods from a project to be deployed to specific nodes.

In the following example, the admission controller evaluates all requests made to a project called **opensource** with the labels **region=west**, **env=test**, **infra=fedora**, and

**os=fedora**. If a pod from that project must be deployed, the pods are scheduled to all nodes with node selector **k3=v3**.

```
admissionConfig:
 pluginConfig:
 PodNodeSelector:
 configuration:
 podNodeSelectorPluginConfig:
 clusterDefaultNodeSelector: "k3=v3"
 opensource: region=west,env=test,infra=fedora,os=fedora
```

## Deploying **Init** Containers

**Init** containers are pods that are deployed sequentially before any application pod is started. They can be used to determine whether or not a pod or service has started before an application pod starts. The **init** container can be used to deploy a Java EE application that accesses a database pod with a database connection pool. By default, a database connection pool connects to the database before the application is started to support a faster response to the application. It is possible that the application pod may start before the database pod, which could cause the application to fail.

To avoid the application start up failure, an **init** container may be created that monitors the database availability before the Java EE pod starts:

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
 app: myapp
 annotations:
 pod.beta.kubernetes.io/init-containers: '[
 {
 "name": "init-mydb",
 "image": "busybox",
 "command": ["sh", "-c", "until nslookup mydb; do echo waiting for mydb; sleep 2; done;"]
 }
]'
spec:
 containers:
 - name: myapp-container
 image: busybox
 command: ['sh', '-c', 'echo The app is running! && sleep 3600']
```

In the previous configuration, the **myapp-container** pod starts whenever the container defined in the **pod.beta.kubernetes.io/init-containers** annotation execution is finished.

## Demonstration: Controlling Startup with **Init** Containers

1. Log in to the **workstation** VM as **student** using **student** as the password.

Run the **demo config-init setup** command to download the required files for this exercise:

```
[student@workstation ~]$ demo config-init setup
```

2. Inspect the YAML definition file that creates the application.

In the **/home/student/D0380/labs/config-init** directory, use a text editor to open the **app-init.yaml** file. This file defines the application that requires a database connection to work:

```
apiVersion: v1
kind: Pod
metadata:
 name: myapp-pod
 labels:
 name: myapp-pod
 annotations:
 pod.beta.kubernetes.io/init-containers: '[①
 {
 "name": "init-myservice",
 "image": "busybox",
 "command": ["sh", "-c", "until nc mysql.apps.lab.example.com 30306; do
 echo waiting for mysql sleep 2; done;"] ②
 }
]'
spec:
 containers:
 - name: myapp
 image: registry.lab.example.com:5000/openshift/hello-openshift③
 ports:
 - name: myapp
 containerPort: 8080

apiVersion: v1
kind: Service④
metadata:
 name: myapp-service
spec:
 ports:
 - name: 8080-tcp
 port: 8080
 protocol: TCP
 targetPort: 8080
 nodePort: 30080
 name: myapp
 selector:
 name: myapp-pod
 type: NodePort

apiVersion: v1
kind: Route⑤
metadata:
 name: myapp-route
 labels:
 name: myapp-pod
spec:
 host: cluster.apps.lab.example.com
 to:
 kind: Service
```

```
name: myapp
```

- ➊ Defines an **init** container that is executed to check that another pod is available.
  - ➋ Runs a **busybox** container (a lightweight container that supports basic networking tools) to make diagnostic checks. In this container, a loop runs the **nc** command to connect to a MySQL container. If that fails, then it reruns the command every two seconds until there is a response from the container.
  - ➌ The **hello-openshift** pod executes after the **init** container is started.
  - ➍ A service to allow external access to the **hello-openshift** container.
  - ➎ A route to access the application with a URL.
3. Deploy the application pod with the **init** container.

Log in to OpenShift as the **developer** user with **redhat** as the password:

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://console.lab.example.com
```

Create a project in OpenShift to deploy the application:

```
[student@workstation ~]$ oc new-project config-init
```

Deploy the application by creating the resources defined in the **/home/student/D0380/labs/config-init/app-init.yaml** file:

```
[student@workstation ~]$ oc create -f /home/student/D0380/labs/config-init/app-
init.yaml
```

4. Inspect the deployment of the pod.

On the **workstation** VM, check that pod is not ready because the **init** container is running:

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
myapp-pod 0/1 Init:0/1 0 4s
```

5. Evaluate the logs generated by the **init** container.

Run the **oc logs** command to evaluate the **init** container output. The **init** container is called **init-myservice**:

```
[student@workstation ~]$ oc logs myapp-pod -c init-myservice
nc: can't connect to remote host (172.25.250.21): No route to host
nc: can't connect to remote host (172.25.250.21): No route to host
waiting for mysql sleep 2
```

The **nc** command tries to connect to the MySQL host but it cannot access it because it was not deployed.

6. Access the application:

```
[student@workstation ~]$ curl cluster.apps.lab.example.com:30080
```

The application was not deployed because the **init** container is still running. The following error page is displayed:

```
curl: (7) Failed connect to cluster.apps.lab.example.com:30080; No route to host
```

7. Deploy the MySQL pod and all associated components to make it visible to the **init** container.

Deploy the MySQL pod by creating the resources defined in the **/home/student/D0380/labs/config-init/mysql-init.yaml** file:

```
[student@workstation ~]$ oc create -f /home/student/D0380/labs/config-init/mysql-init.yaml
```

8. Inspect the deployment of the pod.

On the **workstation** VM, verify that the **mysql-pod** is running because the **init** container was shut down:

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
myapp-pod 0/1 Init:0/1 0 4s
mysql-pod 1/1 Running 0 14s
```

9. Evaluate the logs generated by the **init** container.

Run the **oc logs** command to evaluate the **init** container output. The **init** container is called **init-myservice**.

```
[student@workstation ~]$ oc logs myapp-pod -c init-myservice
...
waiting for mysql sleep 2
```

The **nc** command is not generating errors, and so the **init** container quits.

10. The application was deployed because the **init** container is shut down. Use the **curl** command to access the application:

```
[student@workstation ~]$ curl cluster.apps.lab.example.com:30080
Hello OpenShift!
```

11. Delete the project.

Log in to the cluster as the **admin** user with **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
--insecure-skip-tls-verify=true https://console.lab.example.com
```

Delete the project:

```
[student@workstation ~]$ oc delete project cluster-init
project "config-init" deleted
```

This concludes the demonstration.



## References

Further information is available in the Admission Controllers chapter of the *OpenShift Container Platform Architecture Guide* for OpenShift 3.5 at  
| <https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Configuring Admission Controllers

In this exercise, you will configure a limit on self-provisioned projects to one per user.

## Outcomes

You should be able to configure admission controller plug-ins to limit the number of projects for all users in OpenShift Container Platform.

### Before you begin

The guided exercises from the section called “*Guided Exercise: Executing Pre-installation Tasks*” and the section called “*Guided Exercise: Using the Advanced Installation Method*” must have been completed. If not, complete the guided exercises before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab config-admission setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project. It also downloads the required files for this exercise:

```
[student@workstation ~]$ lab config-admission setup
```

## Steps

1. Update the configuration file responsible for enabling the admission controller.

- 1.1. Connect to the **console** VM from the **workstation** VM.

Use the **ssh** command to log in to the **console** VM as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

- 1.2. Ensure that the **openshift-install** directory, created in the section called “*Guided Exercise: Using the Advanced Installation Method*”, is present in the **/root** directory. If not, use the **git clone** command to retrieve the repository from the **services** VM:

```
[root@console ~]# git clone \
http://services.lab.example.com/openshift-install
```

- 1.3. Inspect the files from the cloned repository.

Go to the **openshift-install/lab.example.com** directory and list the files:

```
[root@console ~]# cd openshift-install/lab.example.com/
[root@console lab.example.com]# ls
hosts-ucf README.md rootCA.crt rootCA.csr rootCA.key
```

- 1.4. Open the master configuration file to limit the number of projects in OpenShift.

---

Use a text editor to open the `/root/D0380/labs/config-admission/master-config.yaml` file. Look for the `admissionConfig` attribute in the YAML file, and identify the `ProjectRequestLimit` admission controller:

```
ProjectRequestLimit:
 configuration:
 apiVersion: v1
 kind: ProjectRequestLimitConfig
 limits:
```

- 1.5. Update the `limits` attribute and set `maxProjects` to 1.

```
limits:
 - maxProjects: 1
```

Save the file.

2. Run the Ansible Playbook that updates the master configuration file.

#### 2.1. Inspect the `/root/D0380/labs/config-admission/admission-setup.yaml` file.

The YAML file is responsible for copying the files from the `console` VM to the master nodes and restarting the services:

```

- hosts: masters
 become: yes
 tasks:
 - name: Backup the master-config.yaml file from the host.
 command:
 mv /etc/origin/master/master-config.yaml /etc/origin/master/master-
 config-8.2.yaml ①

 - name: Copy the master-config.yaml file with the admission controller plug-
 in ②
 copy:
 src: master-config.yaml
 dest: /etc/origin/master/master-config.yaml

 - name: Restart the master API ③
 systemd:
 state: restarted
 name: atomic-openshift-master-api

 - name: Restart the controllers API ④
 systemd:
 state: restarted
 name: atomic-openshift-master-controllers
```

- ① Back up the `master-config.yaml` file.
- ② Update the `master-config.yaml` file on each master node.
- ③ Restart the master API service.
- ④ Restart the master controller's service.

## 2.2. Run the Ansible Playbook to configure the admission controller.

On the **console** VM, run the following commands:

```
[root@console lab.example.com]# cd /root/D0380/labs/config-admission
[root@console config-admission]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf admission-setup.yaml

...output omitted...

PLAY RECAP ****
master1.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master2.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master3.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
```

If any errors are raised, review the **master-config.yaml** for errors.

## 2.3. Log out of the **console** VM.

Run the following command to go back to the **console** VM:

```
[root@console config-admission]# exit
[root@workstation ~]$
```

## 3. Test the configuration change.

### 3.1. On the **workstation** VM, log in as the **developer** user.

Use the **oc login** command to access the master API, using the **developer** user and **redhat** as the password:

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://console.lab.example.com
Login successful.

You don't have any projects. You can try to create a new project, by running
 oc new-project <projectname>
```

### 3.2. Create two projects.

Run the following command to create a new project in OpenShift:

```
[student@workstation ~]$ oc new-project project1
Now using project "project1" on server "https://console.lab.example.com:443".
... output omitted ...
```

Run the command again but with a different project name:

```
[student@workstation ~]$ oc new-project project2
```

This time, an error is raised because the user cannot create more than one project:

---

```
Error from server (Forbidden): projectrequests "project2" is forbidden: user
developer cannot create more than 1 project(s).
```

4. Delete the project.

Run the following command to delete the project:

```
[student@workstation ~]$ oc delete project project1
project "project1" deleted
```

5. Restore the configuration file to the original state.

On the **console** VM, run the following command:

```
[student@workstation ~]$ ssh root@console
[root@console config-admission]# cd /root/D0380/labs/config-admission
[root@console config-admission]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf admission-cleanup.yaml
```

The playbook reverts the original file from each master node.

This concludes the guided exercise.

# Configuring OpenShift for Overcommitment and Idling Services

## Objective

After completing this section, students should be able to configure OpenShift Container Platform to use the overcommitment and idling features.

## Idling Services

Some companies may deploy their environments and applications to a cloud provider. OpenShift Container Platform allows you to manually decrease the number of running pods to zero, but new requests are not processed because the application is down.

OpenShift Container Platform provides an automatic way to restart the application, using the **oc idle** command. Idling an application involves finding the scalable resources (deployment configurations, replication controllers, and others) associated with a service. Idling an application finds the service and marks it as idled, scaling down the resources to zero replicas.

### Idling a Single Service

To idle a single service, OpenShift Container Platform requires a service associated with a pod to become idled. For example:

```
[root@demo ~]# oc idle service-name
```

### Idling Multiple Services

A complex application may use multiple services to start. For example, a database-driven application requires that both the application and the database be idled. To define multiple services, the **--resource-names-file** parameter must refer to a file with each service listed on its own line. For example:

```
[root@demo ~]# oc idle --resource-names-file filename
```

## Requests and Limits

OpenShift Container Platform allows administrators and developers to define the quantity of resources needed by a container or any resource. To customize any container, the following field can be added as part of the pod definition:

```
resources:
 limits:
 cpu: 100m
 memory: 512Mi
```

This information is useful for OpenShift Container Platform to identify the nodes that can be used to deploy the resources. Due to the dynamic nature of OpenShift Container Platform, some resources may be scheduled on a node that is overcommitted. This can happen because OpenShift baselines the currently available quantity of resources on a node. For example, a container may be idle on a node and this resource is not accounted for in the amount of resources used by a node.

## Overcommitting Resources

A node is overcommitted when it has a pod scheduled that makes no request, or when the sum of limits across all pods on that node exceeds available machine capacity. In an overcommitted environment, the pods on the node attempt to use more compute resources than are available. When this occurs, the node gives priority to one pod over another.

Requests and limits enable you to allow and manage the overcommitment of resources on a node, which may be desirable in development environments where a trade-off of guaranteed performance for capacity is acceptable.

## Configuring Master for Overcommitment

The difference between request and limit is the level of overcommitment. Consider the following scenario, where the container has the following memory values:

- Request of 1GiB
- Limit of 2GiB

OpenShift schedules based on the 1GiB request being available on the node, but the container could use up to 2GiB, which means that it is 200% overcommitted.

You can configure master nodes to override the ratio between request and limit set on developer containers.

Configure the **ClusterResourceOverride** admission controller in the **/etc/origin/master/master-config.yaml** file as in the following example:

```
kubernetesMasterConfig:
 admissionConfig:
 pluginConfig:
 ClusterResourceOverride: ①
 configuration:
 apiVersion: v1
 kind: ClusterResourceOverrideConfig
 memoryRequestToLimitPercent: 25 ②
 cpuRequestToLimitPercent: 25 ③
 limitCPUToMemoryPercent: 200 ④
```

- ① The plug-in name.
- ② (Optional, in the range 1-100.) If a container memory limit has been specified or defaulted, the memory request is overridden to this percentage of the limit.
- ③ (Optional, in the range 1-100.) If a container CPU limit has been specified or defaulted, the CPU request is overridden to this percentage of the limit.
- ④ (Optional, positive integer.) If a container memory limit has been specified or defaulted, the CPU limit is overridden to a percentage of the memory limit, with a 100 percentage scaling 1GiB of RAM to equal one CPU core. This is processed prior to overriding CPU request (if configured).



## References

Further information is available in the Idling Applications chapter of the *OpenShift Container Platform Cluster Administration Guide* for OpenShift Container Platform 3.6; at

    | <https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Over-committing chapter of the *OpenShift Container Platform Cluster Administration Guide* for OpenShift Container Platform 3.6; at

    | <https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Idling a Service

In this exercise, you will idle a service that is exposed.

## Outcomes

You should be able to setting up a service as idle.

## Before you begin

The guided exercise from *the section called “Guided Exercise: Executing Pre-installation Tasks”* and *the section called “Guided Exercise: Using the Advanced Installation Method”* must have been completed. If not, complete the guided exercises before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab config-idle setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project:

```
[student@workstation ~]$ lab config-idle setup
```

## Steps

1. Deploy the **hello-openshift** application in the **config-idle** project.
  - 1.1. Use the **oc login** to access the master API using the **developer** user and **redhat** as the password:

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://console.lab.example.com
```

- 1.2. Create a project called **config-idle**.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ oc new-project config-idle
```

- 1.3. Deploy the application.

Use the **oc new-app** command to deploy the application:

```
[student@workstation ~]$ oc new-app \
--docker-image=registry.lab.example.com:5000/openshift/hello-openshift \
--insecure-registry=true --name=config-idle
```

- 1.4. Wait until the pod is deployed.

Run the **oc get pods** command until the pod status is changed to **Running**:

```
[student@workstation ~]$ oc get pods
NAME READY STATUS RESTARTS AGE
config-idle-1-x4qbd 1/1 Running 0 1m
```

- 1.5. Use a route to expose the application.

On the **workstation** VM, run the following command to create a route:

```
[student@workstation ~]$ oc expose svc config-idle \
--hostname=config-idle.apps.lab.example.com
route "config-idle" exposed
```

- 1.6. Verify that the application is accessible.

Use the **curl** command to verify that the application is accessible:

```
[student@workstation ~]$ curl config-idle.apps.lab.example.com
Hello OpenShift!
```

2. Idle the application.

- 2.1. Run the command **oc get svc** to identify the service that you need to idle.

```
[student@workstation ~]$ oc get svc
NAME CLUSTER-IP EXTERNAL-IP PORT(S) AGE
config-idle 172.30.78.237 <none> 8080/TCP,8888/TCP 6m
```

- 2.2. Idle the service.

Run the following command to idle the service:

```
[student@workstation ~]$ oc idle config-idle
The service "config-idle/config-idle" has been marked as idled
The service will undle DeploymentConfig "config-idle/config-idle" to 1 replicas
once it receives traffic
DeploymentConfig "config-idle/config-idle" has been idled
```

- 2.3. Verify that no pods are deployed.

Run the following command to verify the running pods:

```
[student@workstation ~]$ oc get pods
No resources found.
```

3. Verify that the application was successfully idled.

Run the **curl** command again to access the application:

```
[student@workstation ~]$ curl config-idle.apps.lab.example.com
Hello OpenShift!
```

OpenShift Container Platform deploys a new pod to support the new request, which explains why the output took some time.

4. To delete the project, run the following command:

---

```
[student@workstation ~]$ oc delete project config-idle
```



## Note

The command may take some time to complete.

This concludes the guided exercise.

# Managing Operating System and OpenShift Resources

## Objectives

After completing this section, students should be able to:

- Manage operating system resource consumption and cluster resources in OpenShift Container Platform.
- Configure pod scheduling in an OpenShift Container Platform cluster.
- Limit resource usage in an OpenShift Container Platform cluster.

## Scheduling Nodes

OpenShift provides three major scheduling approaches to place pods:

1. *Default scheduling*: Addresses the most common use cases needed to schedule pods on nodes.
2. *Advanced scheduling*: Addresses use cases where a pod must be scheduled to run on the same node where another pod is running.
3. *Custom scheduling*: Addresses uses cases where you need different algorithms from the ones provided by OpenShift Container Platform. It also allows changes to pod specifications.

This course only covers the default scheduling approaches.

## Default Scheduling

The default scheduling approach reads data from the pod and tries to find a node that is a good fit based on the configured policies. It does not modify the pod, but instead creates a binding for the pod that ties it to the particular node.

OpenShift Container Platform provides a *generic scheduler* to handle default scheduling tasks. It uses a 3-step operation to select a suitable node to host a pod:

1. *Filter*: The available nodes are filtered based on the specified constraints or requirements. This is done by running each of the nodes through the list of filter functions, called *predicates*.
2. *Prioritize*: Each node passes through a series of priority functions that assign it a score. The scheduler configuration may also accept a positive numeric value, named **weight**, for each priority function. The node score provided by each priority function is multiplied by a weight factor defined by the administrator. These numbers are combined by just adding the scores for each node provided by all the priority functions.
3. *Select*: The nodes are sorted based on their scores and the node with the highest score is selected to host the pod. If multiple nodes have the same high score, one is randomly selected.

### Configuring Default Scheduling

The scheduler policy is defined on the master host in a file called `/etc/origin/master/scheduler.json`. To modify the scheduler policy:

- Edit the scheduler configuration file to set the desired predicates, priority functions, and priority function weights.
- Restart the OpenShift Container Platform master services for the changes to take effect.

## Controlling Scheduling Pods

OpenShift Container Platform distributes pods using the default scheduler mechanism onto defined nodes to minimize problems. To run pods on the same set of nodes, some priorities functions and predicates support an argument named **serviceAffinity** and or **serviceAntiAffinity**.

### Affinity

Affinity at a particular level indicates that all pods that belong to the same service are scheduled onto nodes that belong to the same level. This handles any latency requirements of applications by allowing administrators to ensure that peer pods are not too geographically separated. If no node is available within the same affinity group to host the pod, then the pod is not scheduled.

### Anti Affinity

Anti affinity (or spread) at a particular level indicates that all pods that belong to the same service are spread across nodes that belong to that level. This ensures that the application is well spread for high availability purposes. The scheduler tries to balance the service pods across all applicable nodes as evenly as possible.

In the following excerpt, OpenShift Container Platform defines predicates and the priority functions as part of a **Policy** definition:

```
kind: "Policy"
version: "v1"
predicates:
 - name: "PodFitsResources" ❶
 argument:
 serviceAffinity: ❷
 labels:
 - "zone"
priorities:
 - name: "LeastRequestedPriority" ❸
 weight: 1
 - name: "RackSpread" ❹
 weight: 1
 argument:
 serviceAntiAffinity:
 labels:
 - "rack"
```

- ❶ Determines a fit based on resource availability. The nodes can declare their resource capacities and then pods can specify what resources they require. Fit is based on requested, rather than used resources.
- ❷ Filters out nodes that do not belong to the specified topological level defined by the provided labels.

- ③ Favors nodes with fewer requested resources. It calculates the percentage of memory and CPU requested by pods scheduled on the node, and prioritizes nodes with the highest available/remaining capacity.
- ④ Avoids running on nodes with the same label.

In this setup, the two predicates require that all pods of the same service end up in the same zone (this is a common setup when latency is a common issue or concern) and that the selected node has the resources required to run a given pod. While the priorities specify that pods of the same service should be spread across racks within a zone and to favor nodes with the least requested resources. This is a very common setup where availability is desired and the latency across all the racks is not a concern.



## Note

A list of comprehensive predicates and priority functions is available in the resources section.

## Node Resources

Among all the predicates provided by the default scheduler in OpenShift Container Platform, some of them embed the resource availability evaluation (for example, memory or CPU), such as the **NodeFitsResources** predicate. Internally, it uses runtime execution information such as memory and CPU usage, but it may reserve resources for both system components from the host (`sshd`, `NetworkManager`) and node components (`kubelet`, `kube-proxy`, and Docker registry).

### Configuring Nodes for Allocated Resources

To configure each node with allocated resources, edit the **kubeletArguments** section of the `/etc/origin/node/node-config.yaml` file.

```
kubeletArguments:
 kube-reserved: ❶
 - "cpu=200m, memory=30G"
 system-reserved: ❷
 - "cpu=200m, memory=30G"
```

- ❶ Resources reserved for node components. If the attribute is not declared, OpenShift Container Platform uses all the resources available except the one defined for the system components.
- ❷ Resources reserved for system components. If the attribute is not declared, OpenShift Container Platform uses all the resources available except the one defined for the node components.

## Managing Node Selectors

You can control pods placement with node selectors, in conjunction with node labels. You can assign labels during the advanced installation, or add them after a node installation.

### Defining Cluster-Wide Node Selectors

You can set cluster-wide default node selectors to restrict pod placement to specific nodes. To do so, edit the master configuration file located at `/etc/origin/master/master-config.yaml` on each master node. This is applied to the pods created in all projects without a specified **nodeSelector** value.

The following excerpt shows how to define a node selector:

```
...
projectConfig:
 defaultNodeSelector: "type=user-node,region=east"
...
```

After defining the label, restart the OpenShift Container Platform services:

```
[root@demo ~]# systemctl restart atomic-openshift-master-controller
[root@demo ~]# systemctl restart atomic-openshift-master-api
```

## Handling Resource Errors

OpenShift Container Platform monitors all nodes and identifies resource shortages, such as lack of memory or disk space. Whenever a node lacks sufficient resources, OpenShift Container Platform identifies and minimizes the problem by applying eviction policies on running pods.

Each node may have its own resource limitation. To configure an eviction policy, edit the **/etc/origin/node/node-config.yaml** file.

### Eviction Policies

OpenShift Container Platform implements two types of eviction:

- *Hard eviction*: This policy immediately kills the pods in the node where the resource shortage has been reached.
- *Soft eviction*: This policy reclaims resources until the grace period is exceeded. If no grace period is provided in the node configuration the node produces an error on startup.

### Hard Eviction Policy

To configure a hard eviction policy, the **node-config.yaml** file requires the **eviction-hard** configuration as part of the **kubeletArguments** section.

```
kubeletArguments:
 eviction-hard: ①
 - memory.available<500Mi ②
 - nodefs.available<500Mi ③
```

- ① Configure a hard eviction policy.
- ② Limits the minimum available memory up to 500 MiB.
- ③ Limits the minimum disk availability up to 500 MiB.

### Soft Eviction Policy

To configure a soft eviction policy, the **node-config.yaml** file requires the **eviction-soft** and the **eviction-soft-grace-period configuration** as part of the **kubeletArguments** section.

```
kubeletArguments:
 eviction-soft: ①
```

```

- memory.available<500Mi ②
- nodefs.available<500Mi
eviction-soft-grace-period: ③
- memory.available=1m30s
- nodefs.available=1m30s

```

- ①** Configures a soft eviction policy.
- ②** Defines the minimum quantity of free resources that a node can have.
- ③** Defines the minimum amount of time that a node should wait until the pods are evicted after the first error signal.



## Note

Memory starvation issues are not detectable in OpenShift Container Platform if swap memory is provided.

If an eviction threshold is met, independent of its associated grace period, the node reports a condition indicating that the node is under memory or disk pressure. This prevents the scheduler from scheduling any additional pods on the node while attempts are made to reclaim resources.

## Implementing Pod Eviction

The node reports an out-of-resource condition indicating that the node is under disk pressure or memory pressure if an eviction threshold is met. This process is independent of the grace period associated with soft eviction.

The node ranks pods for eviction by their *quality of service*. For each pod, the following quality of service levels are available:

- **Guaranteed:** Pods that consume the highest amount of the starved resource relative to their request are failed first. If no pod has exceeded its request, the strategy targets the largest consumer of the resource that a node is running out of.
- **Burstable:** Pods that consume the highest amount of the starved resource relative to their request for that resource are failed first. If there are no pods exceeding their requests, the strategy targets the largest consumer of the resource that a node is running out of.
- **BestEffort:** Pods that consume the highest amount of the resource that a node is running out of are failed first.

A **Guaranteed** pod is never evicted, except in the following cases:

- *The services resources limitations are overcommitted:* In this case, the values defined in the **system-reserved** or **kube-reserved** stanzas of the **node-config.yaml** file are insufficient.
- *Only Guaranteed pods exist in the node:* The node evicts a **Guaranteed** pod that least impacts node stability and limits the impact of the unexpected consumption to remaining pods.

The node ranks pods by quality of service. If the node is responding to a lack of available disk space, it ranks pods with a quality of service that consume the greatest amount of local disk space, and evicts those pods first.



## References

Further information is available in the Scheduling chapter of the *OpenShift Container Platform Cluster Administration Guide* for OpenShift Container Platform 3.6; at

  | <https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Allocating Node Resources chapter of the *OpenShift Container Platform Cluster Administration Guide* for OpenShift Container Platform 3.6; at

  | <https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Handling Out of Resource Errors chapter of the *OpenShift Container Platform Cluster Administration Guide* for OpenShift Container Platform 3.6; at

  | <https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Managing Operating System and OpenShift Resources

In this exercise, you will configure scheduling capabilities on OpenShift Container Platform to deploy pods to a specific node and check that pods do not violate resource limitations on a node.

## Outcomes

You should be able to deploy labeled pods to specific nodes.

## Before you begin

The guided exercises from *the section called “Guided Exercise: Executing Pre-installation Tasks”* and *the section called “Guided Exercise: Using the Advanced Installation Method”* must have been completed. If not, complete the guided exercises before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab config-schedule setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project. It also downloads the required files for this exercise:

```
[student@workstation ~]$ lab config-schedule setup
```

## Steps

1. Update the scheduler configuration file.

- 1.1. Connect to the **console** VM from the **workstation** VM.

Use the **ssh** command to log in to the **console** VM as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

- 1.2. Ensure that the **openshift-install** directory, created in *the section called “Guided Exercise: Using the Advanced Installation Method”* is present in the **/root** directory. If not, use the **git clone** command to retrieve the repository from the **services** VM:

```
[root@console ~]# git clone \
http://services.lab.example.com/openshift-install
```

- 1.3. Inspect the files from the cloned repository.

Go to the **openshift-install/lab.example.com** directory and list the files:

```
[root@console ~]# cd openshift-install/lab.example.com/
[root@console lab.example.com]# ls
hosts-ucf README.md rootCA.crt rootCA.csr rootCA.key
```

- 1.4. Edit the **scheduler.json** configuration file to define a scheduling policy where pods with a label are deployed to specific nodes.



## Note

The following code extract is available in **/root/D0380/labs/config-schedule/predicate.txt** file.

Use a text editor to open the **/root/D0380/labs/config-schedule/scheduler.json** file. Look for the **predicates** section in the JSON file and add the following definition to the end of the section:

```
{
 "name": "Geography",
 "argument": {
 "serviceAffinity": {
 "labels": [
 "geo"
]
 }
 }
}
```

In this configuration, you create a predicate called **Geography** that looks for a label called **geo** for each deployed pod. If that label exists, the scheduler deploys the pod to a node with the same label.

Save the file.

2. Run the Ansible Playbook that updates the scheduler configuration file.
  - 2.1. Inspect the **/root/D0380/labs/config-schedule/schedule-setup.yaml** file.

The YAML file copies the files from the **console** VM to the master nodes and restarts the services:

```

- hosts: masters
 become: yes
 tasks:
 - name: Backup the scheduler.json file from the host.
 command:
 mv /etc/origin/master/scheduler.json /etc/origin/master/
scheduler-8.6.json ①

 - name: Copy the master-config.yaml file with the admission controller plugin
 copy:
 ②
 src: scheduler.json
 dest: /etc/origin/master/scheduler.json

 - name: Restart the master API
 systemd:
 ③
 state: restarted
 name: atomic-openshift-master-api

 - name: Restart the controllers API
```

```
systemd:④
 state: restarted
 name: atomic-openshift-master-controllers
```

- ① Back up the **master-config.yaml** file.
- ② Update the **master-config.yaml** file on each master node.
- ③ Restart the master API service.
- ④ Restart the master controller service.

## 2.2. Run the Ansible Playbook to configure the admission controller.

On the **console** VM, run the following command:

```
[root@console config-schedule]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf schedule-setup.yaml

...output omitted...

PLAY RECAP ****
master1.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master2.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master3.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
```

If any errors are raised, review the **scheduler.json** file for errors.

## 2.3. Log out of the **console** VM.

Run the following command to go back to the **workstation** VM:

```
[root@console config-schedule]# exit
[student@workstation ~]$
```

## 3. Add labels to the nodes to force all the pods to run on a unique node.

### 3.1. On the **workstation** VM, log in as the **admin** user.

Access the master API with the **oc login** command using **admin** as the user and **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

### 3.2. Update the **apps1** node label.

The pods with the **geo=na** label must be scheduled to deploy to the **apps1.lab.example.com** node.

Run the following command:

```
[student@workstation ~]$ oc label node apps1.lab.example.com geo=na
```

### 3.3. Update the **apps2** node label.

---

The pods with the **geo=apac** label must be scheduled to deploy to the **apps2.lab.example.com** node.

Run the following command:

```
[student@workstation ~]$ oc label node apps2.lab.example.com geo=apac
node "apps2.lab.example.com" labeled
```

4. Deploy the application pod.

To evaluate the memory eviction policy, deploy a deployment configuration with a pod that uses the **stress** command. The application pod consumes 512 MiB of memory in two separate threads, a total of 1024 MiB of memory. This amount of memory allocation allows only a single pod to run on the **appsX.lab.example.com** nodes. To ensure that only one node is stressed, the pods use a **nodeSelector** attribute with the **geo=na** label.

4.1. From the **workstation** VM, log in as the **developer** user.

Access the master API with the **oc login** command as the **developer** user and using **redhat** as the password, to avoid changing the default project configuration:

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://console.lab.example.com
```

4.2. Create a new project to deploy the application.

Open a new terminal window to access the **workstation** VM, and run the following command:

```
[student@workstation ~]$ oc new-project config-schedule
```

4.3. Inspect the deployment configuration that deploys the pods to a node. The deployment configuration is available on the **workstation** VM, in the **/home/student/D0380/labs/config-schedule/stress-dc.yaml** file.

```
spec:
 containers:
 - name: "stress"
 image: "programm/stress:latest"
 command: ["/usr/bin/stress", "--verbose"]
 args: ["--vm-bytes","256M" ①,"--cpu","2","--io","1","--vm","2" ②,"--
timeout","1d"]
 nodeSelector:
 geo: "na" ③
```

- ① Allocates 256 MiB of memory for each thread.
- ② Number of threads started.
- ③ Node selector to where all the pods are deployed.

4.4. Deploy the deployment configuration definition files.

On the **workstation** VM, run the following commands:

```
[student@workstation ~]$ cd /home/student/D0380/labs/config-schedule
[student@workstation config-schedule]$ oc create -f stress-dc.yaml
deploymentconfig "stress" created
```

4.5. Increase the number of running pods by setting the replicas.

Run the following command:

```
[student@workstation config-schedule]$ oc scale dc/stress --replicas=3
deploymentconfig "stress" scaled
```

This command starts three instances of the stress application pod.

4.6. Verify the number of pods started:

```
[student@workstation config-schedule]$ oc get pods
```

Because all the nodes have only 2 GiB memory available, and some processes are already consuming resources, only three pods are running.

4.7. Identify the running pods.

Run the following command:

```
[student@workstation config-schedule]$ oc get pods
NAME READY STATUS RESTARTS AGE
stress-1-7p5t9 1/1 Running 0 19m
stress-1-jg5qx 1/1 Running 0 18m
stress-1-m97fn 1/1 Running 0 18m
```



## Note

Eventually one of the pods may run into an **Error** state. This is expected if your VM is short of memory.

4.8. Increase the amount of memory consumed by one of the pods.

From the previous step, get the name of one of the pods and open a shell. Run the **stress** command to increase memory usage

```
[student@workstation config-schedule]$ oc exec -it stress-1-jg5qx \
"/bin/sh"
$ /usr/bin/stress --verbose --cpu 2 --io 1 --vm 1 --vm-bytes 5096M --timeout 1d
stress: info: [15] dispatching hogs: 2 cpu, 1 io, 1 vm, 0 hdd
stress: dbug: [15] using backoff sleep of 12000us
stress: dbug: [15] setting timeout to 86400s
stress: dbug: [15] --> hogcpu worker 2 [16] forked
stress: dbug: [15] --> hogio worker 1 [17] forked
stress: dbug: [15] --> hogvm worker 1 [18] forked
```

...



## Note

The following message is expected:

```
...
stress: FAIL: [24] (452) failed run completed in 2s
```



## Important

Do not run the command more than once, because it triggers a memory intensive application and it may crash the pod.



## Important

Leave the command running and open a new terminal for the following steps.

### 4.9. Evaluate the behavior of the node.

On the **workstation** VM, run the following commands:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
...output omitted...
[student@workstation ~]$ oc describe node apps1.lab.example.com
...output omitted...
6m 6m 1 kubelet, apps1.lab.example.com
Warning SystemOOM System OOM encountered
```

The command logs the issues identified due to the stress condition.

Because the pod is consuming more memory from the node, OpenShift cluster logs the **SystemOOM** error.

### 5. Clean up.

#### 5.1. Delete the project.

Run the following command to delete the project:

```
[student@workstation ~]$ oc delete project config-schedule
```

#### 5.2. As the **admin** user, delete the labels on each node.

On the **workstation** VM, run the following commands to remove the label:

```
[student@workstation ~]$ oc label node apps1.lab.example.com geo-
node "apps1.lab.example.com" labeled
```

```
[student@workstation ~]$ oc label node apps2.lab.example.com geo-
node "apps2.lab.example.com" labeled
```

6. Restore the configuration file to the original state.

Use the **ssh** command to go back to the **console** VM and run the following command:

```
[root@console ~]# cd DO380/labs/config-schedule/
[root@console config-schedule]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf schedule-cleanup.yaml
```

The Ansible Playbook reverts the original file from each master node.

This concludes the guided exercise.

# Lab: Managing System Resources

In this lab, you will configure OpenShift Container Platform to allow developers to create two projects and deploy an application in each project to different nodes.

## Outcomes

You should be able to:

- Set up admission controllers to limit the number of projects created for each developer in OpenShift Container Platform.
- Set up the scheduler to deploy applications to a specific node in OpenShift.

## Before you begin

The guided exercise from chapter 4 must have been completed. If not, complete the guided exercise before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab config-review setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project:

```
[student@workstation ~]$ lab config-review setup
```

## Steps

1. Update the configuration file responsible for limiting the number of projects to two for each user.
  - 1.1. Connect to the **console** VM from the **workstation** VM.
  - 1.2. Ensure that the **openshift-install** directory, created in the guided exercise *Using the Advanced Installation Method* is present in the **/root** directory.
  - 1.3. Update the configuration file with the admission controller to limit the number of projects to two in the OpenShift cluster. The **master-config.yaml** file is available as one of the downloaded files in the **/root/D0380/labs/config-review**.



## Note

Ensure you use correct indentation.

Locate the **limits** attribute and set the value of **maxProjects** to 2.

2. Run the **/root/D0380/labs/config-review/admission-review-setup.yaml** Ansible Playbook with the **/root/openshift-install/lab.example.com/hosts-ucf** Ansible inventory file to update the admission controller.
3. Test the configuration change.
  - 3.1. From the **workstation** VM, log in as the **developer** user in the master API. The password is **redhat**.

3.2. Create three projects: **project-review1**, **project-review2**, and **project-review3**. The third project should fail.

The commands create two new projects in OpenShift. The third one fails.

4. Log in as the **admin** user to the master API using **redhat** as the password:
5. Identify the node label configured during the lab setup process. The label uniquely identifies the node used for scheduling purposes. Only the **apps1.lab.example.com** and **apps2.lab.example.com** are updated.

There is a label named **destination=ny** for this node.

6. Update the scheduler configuration file to deploy the pods to different nodes. Deploy the **mysql** and **hello-openshift** pods to **apps1.lab.example.com** and **apps2.lab.example.com** respectively.
7. On the **console** VM, run the **/root/D0380/labs/config-review/schedule-review-setup.yaml** Ansible Playbook that updates the scheduler configuration file.
8. Test the configuration change.
  - 8.1. From the **workstation** VM, edit the **hello-openshift** pod definition file located at **/home/student/D0380/labs/config-review/review-app-init.yml** to schedule the **apps1.lab.example.com** node. Deploy the template in the **project-review1** project.
  - 8.2. Edit the **mysql** pod definition file called **review-db-init.yml** to schedule the **apps2** node. Deploy the template to the **project-review2** project.
  - 8.3. From the **workstation** VM, log in to the master API as the **developer** user.
  - 8.4. Use the **project-review1** project to deploy the application defined in the **review-app-init.yml** file.
  - 8.5. Use the **project-review2** project to deploy the application defined in the **review-db-init.yml** file.
  - 8.6. Evaluate the placement of the pods.
9. Grade your work.

From the **workstation** VM, run the following command:

```
[student@workstation ~]$ lab config-review grade
```

10. Clean up by restoring the configuration file to the original state.

On the **console** VM, run the following command:

```
[root@console ~]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf \
/root/D0380/labs/config-review/review-cleanup.yaml
```

---

The Ansible Playbook reverts the original file from each master node.

This concludes this lab.

## Solution

In this lab, you will configure OpenShift Container Platform to allow developers to create two projects and deploy an application in each project to different nodes.

### Outcomes

You should be able to:

- Set up admission controllers to limit the number of projects created for each developer in OpenShift Container Platform.
- Set up the scheduler to deploy applications to a specific node in OpenShift.

### Before you begin

The guided exercise from chapter 4 must have been completed. If not, complete the guided exercise before running this lab.

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab config-review setup** command. The script ensures that the cluster is running and that the registry and router pods are running in the **default** project:

```
[student@workstation ~]$ lab config-review setup
```

### Steps

1. Update the configuration file responsible for limiting the number of projects to two for each user.
  - 1.1. Connect to the **console** VM from the **workstation** VM.

Use the **ssh** command to log in to the **console** VM as the **root** user:

```
[student@workstation ~]$ ssh root@console
[root@console ~]#
```

- 1.2. Ensure that the **openshift-install** directory, created in the guided exercise *Using the Advanced Installation Method* is present in the **/root** directory.

If not, run the **git clone** command to retrieve the repository from the **services** VM:

```
[root@console ~]# git clone \
http://services.lab.example.com/openshift-install
```

- 1.3. Update the configuration file with the admission controller to limit the number of projects to two in the OpenShift cluster. The **master-config.yaml** file is available as one of the downloaded files in the **/root/D0380/labs/config-review**.



### Note

Ensure you use correct indentation.

Use a text editor to open the `/root/D0380/labs/config-review/master-config.yaml` file. Look for the `admissionConfig` attribute in the YAML file, and append the `ProjectRequestLimit` admission controller.

Locate the `limits` attribute and set the value of `maxProjects` to 2.

```
ProjectRequestLimit:
 configuration:
 apiVersion: v1
 kind: ProjectRequestLimitConfig
 limits:
 - maxProjects: 2
```

Save the file.

2. Run the `/root/D0380/labs/config-review/admission-review-setup.yaml` Ansible Playbook with the `/root/openshift-install/lab.example.com/hosts-ucf` Ansible inventory file to update the admission controller.

On the **console** VM, run the following commands:

```
[root@console ~]# cd /root/D0380/labs/config-review
[root@console config-review]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf admission-review-setup.yaml

...output omitted...

PLAY RECAP ****
master1.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master2.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master3.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
```

If any errors are raised, review the `master-config.yaml` for errors.

3. Test the configuration change.
- 3.1. From the **workstation** VM, log in as the **developer** user in the master API. The password is **redhat**.

Access the master API with the `oc login` command using the **developer** user.

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://console.lab.example.com
```

- 3.2. Create three projects: **project-review1**, **project-review2**, and **project-review3**. The third project should fail.

Run the following commands:

```
[student@workstation ~]$ oc new-project project-review1
[student@workstation ~]$ oc new-project project-review2
[student@workstation ~]$ oc new-project project-review3
```

The commands create two new projects in OpenShift. The third one fails.

This time, an error is raised because the user cannot create more than one project:

```
Error from server (Forbidden): projectrequests "project-review3" is forbidden:
user developer cannot create more than 2 project(s).
```

4. Log in as the **admin** user to the master API using **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

5. Identify the node label configured during the lab setup process. The label uniquely identifies the node used for scheduling purposes. Only the **apps1.lab.example.com** and **apps2.lab.example.com** are updated.

Run the following command:

```
[student@workstation ~]$ oc describe node apps1.lab.example.com
```

There is a label named **destination=ny** for this node.

Run the following command:

```
[student@workstation ~]$ oc describe node apps2.lab.example.com
```

There is a label named **destination=ca** for this node. Therefore, the label **destination** must be used for scheduling purposes.

6. Update the scheduler configuration file to deploy the pods to different nodes. Deploy the **mysql** and **hello-openshift** pods to **apps1.lab.example.com** and **apps2.lab.example.com** respectively.

Open a terminal window to the **console** VM. Use a text editor to open the **/root/D0380/labs/config-review/scheduler.json** file. Look for the **predicates** section in the JSON file and add the following definition to the end of the section:

```
'
 {
 "argument": {
 "serviceAffinity": {
 "labels": [
 "destination"
]
 }
 },
 "name": "State"
 }
```

Save the file.

7. On the **console** VM, run the **/root/D0380/labs/config-review/schedule-review-setup.yaml** Ansible Playbook that updates the scheduler configuration file.

Run the Ansible Playbook to configure the admission controller using the `/root/openshift-install/lab.example.com/hosts-ucf` Ansible inventory file.

On the **console** VM, run the following command:

```
[root@console ~]# cd /root/D0380/labs/config-review
[root@console config-review]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf schedule-review-setup.yaml
...output omitted...

PLAY RECAP ****
master1.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master2.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
master3.lab.example.com : ok=4 changed=3 unreachable=0 failed=0
```

## 8. Test the configuration change.

- 8.1. From the **workstation** VM, edit the **hello-openshift** pod definition file located at `/home/student/D0380/labs/config-review/review-app-init.yml` to schedule the `apps1.lab.example.com` node. Deploy the template in the **project-review1** project.

To schedule the pod on the **apps1** node, add the following label definition to the **nodeSelector** attribute in the `review-app-init.yml` file:

```
nodeSelector:
 destination: ny
```

Indent the **destination: ny** property by two spaces. Save the file.

- 8.2. Edit the **mysql** pod definition file called `review-db-init.yml` to schedule the **apps2** node. Deploy the template to the **project-review2** project.

To schedule the pod on the **apps2** node, add the following label definition to the **nodeSelector** attribute in the `review-db-init.yml` file:

```
nodeSelector:
 destination: ca
```

Save the file.

- 8.3. From the **workstation** VM, log in to the master API as the **developer** user.

Access the master API with the `oc login` command using the **developer** user and **redhat** as the password, to avoid changing the default project configuration:

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://master1.lab.example.com
```

- 8.4. Use the **project-review1** project to deploy the application defined in the `review-app-init.yml` file.

On the **workstation** VM, run the following commands:

```
[student@workstation ~]$ oc project project-review1
...output omitted...
[student@workstation ~]$ oc create \
-f /home/student/D0380/labs/config-review/review-app-init.yml
...output omitted...
```

- 8.5. Use the **project-review2** project to deploy the application defined in the **review-db-init.yml** file.

On the **workstation** VM, run the following commands:

```
[student@workstation ~]$ oc project project-review2
...output omitted...
[student@workstation ~]$ oc create \
-f /home/student/D0380/labs/config-review/review-db-init.yml
...output omitted...
```

- 8.6. Evaluate the placement of the pods.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ oc get pods -o wide -n project-review1
NAME READY STATUS RESTARTS AGE IP NODE
myapp-pod 1/1 Running 0 4m 10.131.2.2
apps1.lab.example.com
```

From the output, you can see that the MySQL pod was deployed to the **apps1** VM. This is expected because the scheduler was configured to look for that label.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ oc get pods -o wide -n project-review2
NAME READY STATUS RESTARTS AGE IP NODE
mysql-pod 1/1 Running 0 4m 10.131.2.2
apps2.lab.example.com
```

From the output, you can see that the MySQL pod was deployed to the **apps2** VM. This is expected because the scheduler was configured to look for that label.

9. Grade your work.

From the **workstation** VM, run the following command:

```
[student@workstation ~]$ lab config-review grade
```

10. Clean up by restoring the configuration file to the original state.

On the **console** VM, run the following command:

```
[root@console ~]# ansible-playbook \
...
```

```
-i /root/openshift-install/lab.example.com/hosts-ucf \
/root/D0380/labs/config-review/review-cleanup.yaml
```

The Ansible Playbook reverts the original file from each master node.

This concludes this lab.

## Summary

In this chapter, you learned:

- OpenShift Container Platform provides admission controllers to intercept requests from master API users to configure the runtime environment, such as proxy configurations and permissions.
- The **ProjectRequestLimit** admission controller can be used to limit project creation in OpenShift Container Platform.
- Any pod deployed to an OpenShift Container Platform cluster that consumes resources without any need is not desirable due to costs. Use the **oc idle** command to idle the application.
- OpenShift implements three scheduling approaches to deploying pods: default, custom, and advanced.
- The **scheduler.json** file defines the scheduling and priority functions that a node uses to deploy pods.
- OpenShift has mechanisms to evict pods whenever a node runs out of resources.
- There are two approaches to evicting pods:
  - Hard eviction: Pods are immediately evicted whenever a shortage occurs.
  - Soft eviction: Pods are evicted after a resource shortage has existed for a certain amount of time.



## CHAPTER 9

# CONFIGURING SECURITY PROVIDERS AND ADVANCED SECURITY OPTIONS

| Overview          |                                                                                                                                                                                                                                                                                                         |
|-------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Goal</b>       | Configure security providers and advanced security options.                                                                                                                                                                                                                                             |
| <b>Objectives</b> | <ul style="list-style-type: none"><li>Describe the security providers supported by OpenShift Container Platform.</li><li>Configure external security providers using the advanced installation method.</li><li>Configure security settings that affect project creation and user permissions.</li></ul> |
| <b>Sections</b>   | <ul style="list-style-type: none"><li>Describing OpenShift Security Providers (and Quiz)</li><li>Configuring External Security Providers (and Guided Exercise)</li><li>Configuring User Security Options (and Guided Exercise)</li></ul>                                                                |
| <b>Lab</b>        | Configuring Security Providers and Advanced Security Options                                                                                                                                                                                                                                            |

# Describing OpenShift Container Platform Security Providers

## Objective

After completing this section, students should be able to describe the security providers supported by OpenShift Container Platform.

## Introduction to Users and Groups

OpenShift Container Platform was designed from the beginning with scalability and interoperability in mind. This includes how users are managed and authenticated, and how tokens are granted. The authentication layer identifies incoming authentication requests and takes the appropriate action, such as validating the users, identifying the groups that a user belongs, and granting tokens.

You can configure users and groups in OpenShift Container Platform to enable individually or as a group:

### Users

A user is a logical entity that can make requests to the OpenShift API. For example, you can create a **developer** user for the developers in the organization, and an **administrator** user for managing the cluster.

### Groups

You can assign users to one or more groups, where each group has a specific set of permissions that are applied to all members. For example, you can instruct the OpenShift cluster to allow full access to certain API objects to all users in the **api-users** group.

### Note

In addition to the groups that you define, OpenShift Container Platform uses *system groups*, or *virtual groups*, which are automatically provisioned by the platform.

The **system:authenticated** group is automatically associated with any user that is currently authenticated. The **system:unauthenticated** group is automatically associated with any user that is not authenticated.

## Introduction to Security Providers

Security providers allow you to manage in a pluggable fashion the authentication of users in OpenShift Container Platform. This authentication platform is implemented via a built-in OAuth server, which acts as a gateway between users and the cluster. It manages authentication via a set of *identity providers*, which are used to determine the identity of the user making a request. The OAuth server creates an access token, and returns it for authentication use. OAuth in OpenShift Container Platform supports the following identity providers:

- HTPasswd: Uses the default Apache authentication system, which stores credentials in text files.

- Keystone: Uses the Red Hat OpenStack Platform identity service, Keystone.
- LDAP: Uses an LDAP server for managing users.
- Basic authentication: Uses a generic remote authentication server for authenticating users.
- GitHub and GitLab: Uses the OAuth integration that the two providers support for authenticating users.
- Google and OpenID: Uses the OpenID authentication framework for authenticating users.

To set up and configure the security provider, update the master node configuration file, available at `/etc/origin/master/master-config.yaml`. Either the quick and advanced installation methods can be used for configuring an identity provider. They use the **Deny All** method by default, which denies access for all user names and passwords.

When running a master node without a configuration file, the **Allow All** method is used, which allows anyone to log in.



## Important

While the **Allow All** method is useful for testing purposes, this is not a recommended method for a production environment.



## Note

After updating the identity provider, you need to restart the master service:

```
[root@demo ~]# systemctl restart atomic-openshift-master
```

In a cluster, you need to restart the following service:

```
[root@demo ~]# systemctl restart atomic-openshift-master-api
```

## Configuring Identity Providers with Ansible

The default authentication setting is overridden when using the **openshift\_master\_identity\_providers** parameter in the advanced installation method. The following example shows how to use the HTPasswd identity provider in the Ansible inventory file:

```
openshift_master_identity_providers=\
[{'name': 'htpasswd_auth', \
'login': 'true', 'challenge': 'true', \
'kind': 'HTPasswdPasswordIdentityProvider', \
'filename': '/etc/origin/master/htpasswd'}]
①
openshift_master_htpasswd_users=\
{'admin': '$apr1$4ZbKL26l$3eKL/6AQM80941RwTAu611', \
'developer': '$apr1$4ZbKL26l$3eKL/6AQM80941RwTAu611'}②
```

- ① Sets the identity provider to HTPasswd.
- ② Creates the **admin** and **developer** users, which are saved in the **/etc/origin/master/htpasswd** file.

The following table lists some of the available identity providers that you can set in the Ansible inventory file. Consult the references for a detailed list of supported providers.

| Identity Providers for Inventory Files   |                         |
|------------------------------------------|-------------------------|
| Variable                                 | Provider                |
| <b>HTPasswdPasswordIdentityProvider-</b> | HTPasswd                |
| <b>AllowAllPasswordIdentityProvider</b>  | Allow All               |
| <b>LDAPPasswordIdentityProvider</b>      | LDAP                    |
| <b>BasicAuthPasswordIdentityProvider</b> | Basic Authentication    |
| <b>KeystonePasswordIdentityProvider</b>  | Keystone Authentication |
| <b>GitHubIdentityProvider</b>            | GitHub Authentication   |

## Configuring Identity Providers

To configure the identity provider to use, modify the master node configuration file. The following table describes the four parameters common to all identity providers:

| Identity Provider Configuration Options |                                                                                                                                                                                                                                                                                                               |
|-----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Parameter                               | Description                                                                                                                                                                                                                                                                                                   |
| <b>name</b>                             | Provides a generic name for the identity provider, for example, <b>htpasswd_auth</b> .                                                                                                                                                                                                                        |
| <b>challenge</b>                        | When set to <b>true</b> , any unauthenticated token requests from non-web clients, such as the command-line interface, are sent a <b>WWW-Authenticate</b> challenge header.<br><br>This option is supported only to authenticate systems such as <b>etcd</b> and OpenShift Container Platform authentication. |
| <b>login</b>                            | When set to <b>true</b> , any unauthenticated token requests from web clients are redirected to a login page backed by the configured provider. This option is supported only to authenticate users.                                                                                                          |
| <b>mappingMethod</b>                    | This option defines how new identities are mapped to users when they log in to OpenShift Container Platform.                                                                                                                                                                                                  |

The following excerpt shows the configuration of the **HTPasswd** identity provider in the master file:

```
identityProviders:
 - challenge: true①
 login: true②
 mappingMethod: claim③
 name: htpasswd_auth
 provider:
 apiVersion: v1
```

```
file: /etc/origin/master/htpasswd
kind: HTPasswdPasswordIdentityProvider④
```

- ① Enables the challenge header for non-web clients.
- ② Redirects the authentication request to a login page, as defined by the identity provider.
- ③ Defines how identities are mapped to users.
- ④ Defines the identity provider.

The **mappingMethod** property defines how identities are mapped to providers. For example, when the **mappingMethod** is set to **claim**, as listed previously, OAuth fails if the identity is mapped to a previously existing user name.

The following table describes the supported values for the **mappingMethod** property:

| <b>mappingMethod Options</b> |                                                                                                                                                                                                                                                                                                                                                        |
|------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Parameter</b>             | <b>Description</b>                                                                                                                                                                                                                                                                                                                                     |
| <b>claim</b>                 | This option provisions users with the identity's defined user name. The authentication fails if the user name is already mapped to another identity.                                                                                                                                                                                                   |
| <b>lookup</b>                | This option looks for an existing user, but it does not automatically provision users. This allows you to set up identities and users manually, or using an external process.                                                                                                                                                                          |
| <b>generate</b>              | This option provisions a user name with the identity's defined user name. If the user is already mapped to an existing identity, a unique user name is generated. This method should not be using in combination with external processes that require an exact match between OpenShift Container Platform user names and identity provider user names. |
| <b>add</b>                   | This option provisions a user with the identity's defined user name. If a user with that user name exists, the identity is mapped to the existing user. This option is required when multiple identity providers are configured that identify the same set of users and map to the same user names.                                                    |

### Basic Authentication

Set the **identityProviders** attribute to **BasicAuthPasswordIdentityProvider** to validate user names and passwords against a remote server using a server-to-server basic authentication request over HTTP or HTTPS. With the basic authentication, user names and passwords are validated against a remote URL, which is protected by a basic authentication and returns a JSON-formatted response.

A return code of **401** indicates a failed authentication, a return code different from **200** indicates an error, and a return code of **200** indicates a successful authorization.

When the response returns a successful authentication, it may also provide additional data, such as:

- A display name:

```
{"sub":
```

```
"userid",
 "name": "User Name",
...
}
```

- An email address:

```
{"sub":
 "userid",
 "email": "user@demo.example.com",
...
}
```

- A preferred user name. This field is useful when the user identifier is a database key or UUID:

```
{"sub": "A7EFC37-4F0E-4D6C-8923-9CC093CB9DAD",
"preferred_username": "anonymous-user",
...}
```

The following excerpt shows the master configuration file with basic authentication as the identity provider:

```
oauthConfig:
...
 identityProviders:
 - challenge: true 1
 login: true 2
 mappingMethod: claim 3
 name: remote_basic_auth_provider 4
 provider:
 apiVersion: v1
 kind: BasicAuthPasswordIdentityProvider
 url: https://www.example.com/remote-idp 5
 ca: /path/to/ca.file 6
 certFile: /path/to/client.crt 7
 keyFile: /path/to/client.key 8
```

- 1** Unauthenticated token requests from non-web client are sent a **WWW-Authenticate** challenge header for the provider.
- 2** Unauthenticated token requests from web clients are redirected to a login page backed by the identity provider.
- 3** Uses **claim** as the mapping method.
- 4** Defines the prefix to the returned user ID to form the identity name.
- 5** Defines the URL that accepts credentials in HTTP basic authentication headers. Authentication requests are sent to this address.
- 6** This optional parameter defines the certificate authority to use to validate server certificates.
- 7** This optional parameter defines the certificate to present when making authentication requests.
- 8** Defines the key for the client certificate. This option is mandatory if **certFile** is set.

## Request Header

The **RequestHeaderIdentityProvider** identity provider sets request header as the identity provider. This is used to identify users from request headers, such as the **X-Remote-User** HTTP header. Use this identity provider in combination with an authenticating proxy, such as Apache, which is capable of setting the request header value.

For users to authenticate with the request header method, they must access the **<https://master.demo.example.com/oauth/authorize/sub-path>** URL via an authenticating proxy. This is done by configuring the OAuth server to redirect unauthenticated requests to the proxy endpoint that proxies the **<https://master.demo.example.com/oauth/authorize>** URL.



## Important

If unauthenticated requests reach the OAuth server, the **clientCA** parameter must be set for this identity provider. This checks the incoming requests for a valid client certificate before the request headers are checked for a user name. If no certificate is defined, any direct request to the OAuth server can impersonate any identity from the provider by setting a request header.

The following excerpt shows the master configuration file with request header as the identity provider:

```
oauthConfig:
...
 identityProviders:
 - challenge: true 1
 login: true 2
 mappingMethod: claim 3
 name: request_header_provider 4
 provider:
 apiVersion: v1
 kind: RequestHeaderIdentityProvider
 challengeURL: "https://www.example.com/challenge-proxy/oauth/authorize?${query}" 5
 loginURL: "https://www.example.com/logging-proxy/oauth/authorize?${query}" 6
 clientCA: /path/to/ca.file 7
 clientCommonName: 8
 - my-auth-proxy
 headers: 9
 - X-Remote-User
 - SSO-User
 emailHeaders: 10
 - X-Remote-User-Email
 name-Headers: 11
 - X-Remote-User-Display-Name
 preferredUsernameHeaders: 12
 - X-Remote-User-Login
```

- 1** Unauthenticated token requests from non-web client are sent a **WWW-Authenticate** challenge header for the provider.

- ❷ Unauthenticated token requests from web clients are redirected to a login page backed by the identity provider.
- ❸ Uses **claim** as the mapping method.
- ❹ Defines the prefix to the returned UID to form the identity name.
- ❺ This optional parameter defines the URL to redirect the unauthenticated requests. The URL authenticates browser-based clients before proxying them to **https://master.demo.example.com/oauth/authorize**.

The URL that proxies to the master must end with **/authorize** (without a trailing slash), and must also proxy subpaths to ensure that the OAuth approval workflow works properly. **#{query}** is replaced with the actual query string passed.

- ❻ The target URL for redirected unauthorized requests. The URL authenticates clients that expect **WWW-Authenticate** challenges before proxying them to **https://master.demo.example.com/oauth/authorize**.
- ❼ **#{query}** is replaced with the actual query string passed.
- ❽ This optional parameter defines the PEM-encoded certificate bundle. If set, a valid client certificate must be presented and validated against the certificate authorities in the specified file before the request headers are checked for user names.
- ❾ This optional parameter defines a list of *Common Names* (CN). If set, a valid client certificate with a Common Name must be presented before the request headers are checked for user names. If the parameter is not set, any CN is allowed.



## Note

The option can only be used in combination with the **clientCA** parameter.

- ❿ Defines the header names to check for the user identity. The first header that contains a value is used as the identity. This parameter is case-insensitive.
- ⓫ This optional parameter defines the header names to check for an email address. The first header that contains a value is used as the email address. This parameter is case-insensitive.
- ⓬ This optional parameter defines the header names to check for a display name. The first header that contains a value is used as the display name. This parameter is case-insensitive.
- ⓭ This optional parameter defines the header names to check for a preferred user name, only if it is different from the identity determined from the headers specified in **headers**. The first header containing a value is used as the preferred user name. This parameter is case-insensitive.

### Keystone Authentication

Keystone is the identity service for Red Hat OpenStack Platform. Use the **KeystonePasswordIdentityProvider** setting to use Keystone as the identity provider. OpenShift Container Platform supports Keystone v3.

The following excerpt shows how to use Keystone as the identity provider:

```
oauthConfig:
 ...
 identityProviders:
 - challenge: true①
 login: true②
```

```

mappingMethod: claim③
name: keystone_provider④
provider:
 apiVersion: v1
 kind: KeystonePasswordIdentityProvider
 domainName: default⑤
 url: http://keystone.demo.example.com:5000⑥
 ca: ca.pem⑦
 certFile: keystone.pem⑧
 keyFile: keystoneKey.pem⑨

```

- ①** Unauthenticated token requests from non-web clients are sent a **WWW-Authenticate** challenge header for the provider.
- ②** Unauthenticated token requests from web clients are redirected to a login page backed by the identity provider.
- ③** Uses **claim** as the mapping method.
- ④** Defines the prefix to the returned UID to form the identity name.
- ⑤** The Keystone domain name. In Keystone, user names are specific to a domain, such as **default**. Only a single domain is supported.
- ⑥** Defines the URL of the Keystone server.
- ⑦** This optional parameter defines the certificate bundle to use to validate the server certificate for the Keystone URL.
- ⑧** This optional parameter defines the client certificate to present when making requests to the configured URL.
- ⑨** This parameter defines the key for the client certificate. The parameter is mandatory if **certFile** is used.

### LDAP Authentication

LDAP (*Lightweight Directory Access Protocol*) is a protocol used for accessing and maintaining distributed directories. Use the **identityProviders** attribute with the **LDAPPASSWORDIDENTITYPROVIDER** value as to authenticate users against an LDAP v3 server, using simple bind authentication.

When an authentication request is made to an LDAP server, the directory is searched for an entry that matches the user name. If a single match is found, the server attempts a bind using the *distinguished name (DN)* of the entry plus the password that is provided. The following sequence describes the authentication process:

1. The authentication server generates a search filter that combines the attribute and filter in the configured URL with the provided user name.
2. Use the generated filter to search the LDAP server. If the search does not return an entry, access is denied.
3. The authentication server attempts to bind to the LDAP server using the distinguished name of the entry retrieved from the search, and the password provided with the authentication request.
4. If the bind is unsuccessful, access is denied. If the bind is successful, the authentication server builds an identity using the following attributes, if configured:

- Email address
- Display name
- Preferred user name

The following excerpt shows how to use LDAP as the identity provider:

```
oauthConfig:
...
 identityProviders:
 - challenge: true 1
 login: true 2
 mappingMethod: claim 3
 name: ldap_provider 4
 provider:
 apiVersion: v1
 kind: LDAPPASSWORDIdentityProvider
 attributes:
 id: 5
 - dn
 email: 6
 - mail
 name: 7
 - cn
 preferredUsername: 8
 - uid
 bindDN: "" 9
 bindPassword: "" 10
 ca: my-ldap-ca-bundle.crt 11
 insecure: false 12
 url: "ldap://ldap.example.com/ou=users,dc=acme,dc=com?uid" 13
```

- 1** Unauthenticated token requests from non-web clients are sent a **WWW-Authenticate** challenge header for the provider.
- 2** Unauthenticated token requests from web clients are redirected to a login page backed by the identity provider.
- 3** Uses **claim** as the mapping method.
- 4** Defines the prefix to the returned user ID to form the identity name.
- 5** Defines the list of attributes to use as the identity. The first nonempty attribute is used. At least one attribute is required; if none of the listed attributes have a value, the authentication fails.
- 6** Defines the list of attributes to use as the email address. The first nonempty attribute is used.
- 7** Defines the list of attributes to use as the display name. The first nonempty attribute is used.
- 8** Defines the list of attributes to use as the preferred user name. The first nonempty attribute is used.
- 9** This optional parameter defines the distinguished name to use for binding during the search.

- ⑩ This optional parameter defines the password to use during the search. You can also provide this value in an environment variable, an external file, or an encrypted file.
- ⑪ Defines the certificate bundle to use to validate the server certificates against the configured URL.
- ⑫ When set to **True**, no TLS connection is made to the server. When set to **False**, the URLs that use **ldaps://** use TLS, and the URLs that use **ldap://** are automatically upgraded to TLS.
- ⑬ This parameter defines the LDAP host and search parameters to use.



## Note

Use the **lookup** mapping method to whitelist users for an LDAP integration . You must create an identity and a user object for each LDAP user before you can log in using LDAP.

### GitHub Authentication

You can use GitHub as the identity provider for OpenShift Container Platform. This allows you to manage the users in the organization via the GitHub user management system. To use GitHub as the identity provider, set in the **identityProviders** section of the master configuration file the value **GitHubIdentityProvider** . GitHub authentication uses the OAuth integration.



## Important

Using GitHub as an identity provider allows any GitHub user to authenticate to your server. However, you can use the **organizations** configuration attribute to limit authentication to members of specific GitHub organizations.

The following excerpt shows how to use GitHub as the identity provider:

```

oauthConfig:
...
 identityProviders:
 - challenge: false ①
 login: true ②
 mappingMethod: claim ③
 name: github_provider ④
 provider:
 apiVersion: v1
 kind: GitHubIdentityProvider
 clientID: 0eee49e3fe98a01d8cfb ⑤
 clientSecret: 383989d8295cd319a5b28a4abfafab9640c27900 ⑥
 organizations: ⑦
 - myorganization1/team-a
 - myorganization2/team-b
 teams: ⑧
 - myorganization1/team-a
 - myorganization2/team-b

```

- ① The **GitHubIdentityProvider** cannot be used to send **WWW-Authenticate** challenges.

- ② When set to **true**, unauthenticated token requests from web clients, such as the web console, are redirected to GitHub to log in.
- ③ Uses **claim** as the mapping method.
- ④ The provider name is prefixed to the GitHub numeric user ID to form the identity name. It is also used to build the callback URL.
- ⑤ This setting defines the client ID of a registered GitHub OAuth application. You must configure the application with a callback URL of **http://master/oauth2callback/identityProviderName**.
- ⑥ This setting defines the secret issued by GitHub. Optionally, you can provide the secret in an environment variable, an external file, or an encrypted file.
- ⑦ This optional parameter defines a list of organizations. If specified, only GitHub users that are members of at least one of the listed organizations are allowed to log in. If the GitHub OAuth application configured in the **clientID** attribute is not owned by the organization, an organization owner must grant third-party access in order to use this option. The parameter cannot be used in combination with the **teams** field.
- ⑧ This optional parameter defines a list of teams. If specified, only GitHub users that are members of at least one of the listed teams are allowed to log in. If the GitHub OAuth application configured by the **clientID** setting is not owned by the team's organization, an organization owner must grant third-party access in order to use this option. You can do this during the first GitHub login by the organization's administrator, or from the GitHub organization settings. The parameter cannot be used in combination with the **organizations** field.

### HTPasswd Authentication

The **HTPasswd** authentication method allows the validation of user names against a flat file generated using the **htpasswd** command. OpenShift Container Platform supports the Bcrypt, SHA-1, and MD5 cryptographic hash functions, and MD5 is the default for the HTPasswd method. Plain text, encrypted text, and other hash functions are not currently supported by OpenShift Container Platform. The flat file is reread if its modification time changes, without requiring a server restart.

To set up **HTPasswd** authentication, Use the **htpasswd** command with a user name and a hashed password to create the flat file. The following excerpt shows how to use the **HTPasswd** method for OpenShift Container Platform:

```
oauthConfig:
...
 identityProviders:
 - challenge: true ①
 login: true ②
 mappingMethod: claim ③
 name: htpasswd_provider ④
 provider:
 apiVersion: v1
 kind: HTPasswdPasswordIdentityProvider
 file: /path/to/htpasswd/file ⑤
```

- ① Unauthenticated token requests from non-web clients are sent a **WWW-Authenticate** challenge header for the provider.
- ② Unauthenticated token requests from web clients are redirected to a login page backed by the identity provider.

- ③ Uses **claim** as the mapping method.
- ④ Defines the prefix to the returned user ID to form the identity name.
- ⑤ Defines the path to the file that contains the user names and the hashed passwords.



## References

The OAuth Project  
<https://oauth.net>

Further information is available in the Configuring Authentication And User Agent section of the *OpenShift Container Platform Installation and Configuration Guide* at  
<https://access.redhat.com/documentation/en-US/index.html>

# Quiz: Describing OpenShift Container Platform Security Providers

Choose the correct answers to the following questions:

1. Which of the following files stores the identity provider settings?
  - a. `/etc/origin/identity-providers.yaml`
  - b. `/etc/origin/master/master-config.yaml`
  - c. `/etc/origin/master/providers.yaml`
  - d. `/etc/origin/providers/identity.yaml`
2. Which three of the following providers are supported for authentication by OpenShift Container Platform? (Choose three.)
  - a. HTPasswd
  - b. LDAP
  - c. Kerberos
  - d. Basic Authentication
3. Which of the following parameters is used to redirect unauthenticated token requests to a login page backed by the authentication provider?
  - a. `name`
  - b. `login`
  - c. `challenge`
  - d. `mappingMethod`
4. Which of the following settings defines the identity provider?
  - a. `kind`
  - b. `name`
  - c. `login`
  - d. `challenge`
5. Which of the following `mappingProperty` methods provisions a user name with the identity's defined user name?
  - a. `add`
  - b. `claim`
  - c. `lookup`
  - d. `generate`

## Solution

Choose the correct answers to the following questions:

1. Which of the following files stores the identity provider settings?
  - a. `/etc/origin/identity-providers.yaml`
  - b. `/etc/origin/master/master-config.yaml`
  - c. `/etc/origin/master/providers.yaml`
  - d. `/etc/origin/providers/identity.yaml`
2. Which three of the following providers are supported for authentication by OpenShift Container Platform? (Choose three.)
  - a. `HTPasswd`
  - b. `LDAP`
  - c. `Kerberos`
  - d. `Basic Authentication`
3. Which of the following parameters is used to redirect unauthenticated token requests to a login page backed by the authentication provider?
  - a. `name`
  - b. `login`
  - c. `challenge`
  - d. `mappingMethod`
4. Which of the following settings defines the identity provider?
  - a. `kind`
  - b. `name`
  - c. `login`
  - d. `challenge`
5. Which of the following **mappingProperty** methods provisions a user name with the identity's defined user name?
  - a. `add`
  - b. `claim`
  - c. `lookup`
  - d. `generate`

# Configuring External Security Providers

## Objective

After completing this section, students should be able to configure external security providers using the advanced installation method.

## Declaring External Security Providers in Ansible Inventory Files

OpenShift Container Platform supports multiple security providers. Use the advanced installation method to configure the provider suited to your environment (or organization). If no configuration is set, OpenShift Container Platform uses the **DenyAll** provider, which prevents connection to the OpenShift cluster. To support other providers, use the **openshift\_master\_identity\_providers** variable in the **[OSEv3:vars]** section of the inventory file, as mentioned in the previous section.

The following table shows you how to configure various identity providers.

| Identity Providers for Inventory Files |                                                                                                                                                                                                                                                                                                                                                                                                                                                                           |
|----------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| Variable                               | Configuration                                                                                                                                                                                                                                                                                                                                                                                                                                                             |
| HTPasswd                               | <pre>[{"name": "htpasswd_auth", "login": "true", \ 'challenge': "true", \ 'kind': 'HTPasswdPasswordIdentityProvider', \ 'filename': '/etc/origin/master/htpasswd'}]</pre>                                                                                                                                                                                                                                                                                                 |
| AllowAll                               | <pre>[{"name": "allow_all", "login": "true", \ 'challenge': "true", "kind": 'AllowAllPasswordIdentityProvider'}]</pre>                                                                                                                                                                                                                                                                                                                                                    |
| LDAP                                   | <pre>[{"name": "my_ldap_provider", "challenge": "true", \ 'login': 'true', 'kind': 'LDAPPASSWORDIdentityProvider', \ 'attributes': {'id': ['dn'], 'email': ['mail'], \ 'name': ['cn'], 'preferredUsername': ['uid']}, \ 'bindDN': '', 'bindPassword': '', 'ca': 'my-ldap-ca.crt', \ 'insecure': 'false', \ 'url': 'ldap://ldap.example.com:389/ou=users,dc=example,dc=com?uid'}]</pre>                                                                                    |
| OpenID                                 | <pre>[{"name": "openid_auth", "login": "true", \ "challenge": "false", "kind": "OpenIDIdentityProvider", \ "client_id": "my_client_id", "client_secret": "my_client_secret", \ "claims": {"id": ["sub"], "preferredUsername": \ ["preferred_username"]}, \ "name": ["name"], "email": ["email"]}, \ "urls": {"authorize": "https://myidp.example.com/oauth2/authorize", \ \ "token": "https://myidp.example.com/oauth2/token"}, \ "ca": "my-openid-ca-bundle.crt"}]</pre> |

In an existing OpenShift Container Platform instance, the configuration change must be executed on each master node in the **/etc/origin/master/master-config.yaml** configuration

file. To configure the security provider, update the **identityProviders** section with the parameters presented in the previous section.



### Note

To apply the changes, run the following command to restart the **atomic-openshift-master-api** service:

```
[root@demo ~]# systemctl restart atomic-openshift-master-api
```

## Demonstration: Configuring External Security Providers

1. Log in to the **workstation** VM as **student** using **student** as the password.

Run the **demo sec-ldap setup** command to download the required files for this exercise and install an LDAP client on the **workstation** VM:

```
[student@workstation ~]$ demo sec-ldap setup
```

2. In the existing terminal window, log in to the **console** VM as **root**.

```
[student@workstation ~]$ ssh root@console
```

3. Inspect the **/root/D0380/labs/sec-ldap/sec-ldap-setup.yaml** file:

```

- hosts: masters
 become: yes
 tasks:
 - name: Backup the master-config.yaml file from the host. ①
 command:
 mv /etc/origin/master/master-config.yaml /etc/origin/master/master-
 config-9.1.yaml

 - name: Copy the master-config.yaml file with the LDAP security provider ②
 copy:
 src: master-config.yaml
 dest: /etc/origin/master/master-config.yaml

 - name: Restart the master API ③
 systemd:
 state: restarted
 name: atomic-openshift-master-api
```

- ① The Ansible Playbook backs up the **master-config.yaml** file on all the master nodes.
- ② Updates the **master-config.yaml** file with a different configuration file.
- ③ Restart the OpenShift cluster master API to refresh the changes.

Close the file.

4. Inspect the `/root/D0380/labs/sec-ldap/master-config.yaml` file with LDAP provider configuration.

From the **console** VM, open the `master-config.yaml` configuration file. Look for the `identityProviders` section:

```
identityProviders:
 - challenge: true
 login: true
 mappingMethod: claim
 name: ldap_auth
 provider:
 apiVersion: v1
 kind: LDAPPASSWORDIdentityProvider①
 attributes:
 id: ②
 - dn
 preferredUsername:
 - uid ③
 insecure: true
 url: "ldap://services.lab.example.com:389/ou=people,dc=redhat,dc=com?uid" ④
 bindDN: ""
 bindPassword: ""
```

- ① Configures the LDAP provider as the authentication type.
- ② Identifies where in the LDAP search tree the user name is located.
- ③ Defines the display name to use.
- ④ Points to the host where the LDAP server is available.

5. Inspect the LDAP search.

On the **workstation** VM, run the OpenLDAP client tool to search for entries:

```
[student@workstation ~]$ ldapsearch -x -H ldap://services.lab.example.com:389/ \
-b "ou=people,dc=redhat,dc=com" -s sub
extended LDIF
#
LDAPv3
base <ou=people,dc=redhat,dc=com> with scope subtree
filter: (objectclass=*)
requesting: ALL
#
people, redhat.com
dn: ou=people,dc=redhat,dc=com
ou: people
description: All people in organisation
objectClass: organizationalUnit

admin, people, redhat.com
dn: uid=admin,ou=people,dc=redhat,dc=com
objectClass: top
objectClass: inetOrgPerson
uid: admin
cn: Joe
sn: Admin
userPassword:: YWRtaW4=
```

...

The **ldapsearch** command shows a set of users named **btX** with the password **ldap1**.

- On the **console** VM, run the Ansible Playbook to update the **master-config.yaml** file:

```
[root@console ~]# cd DO380/labs/sec-ldap
[root@console sec-ldap]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf sec-ldap-setup.yaml
```

- Check that the new user can connect to the OpenShift cluster.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ oc login -u bt1 -p ldap1
Login successful.

You don't have any projects. You can try to create a new project, by running
oc new-project <projectname>
```

- Add the cluster administration role to the user.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ ssh root@master1 oc adm policy add-cluster-role-to-user
cluster-admin bt1
cluster role "cluster-admin" added: "bt1"
```

- Revert the changes.

On the **console** VM, run the following command:

```
[root@console sec-ldap]# ansible-playbook -i /root/openshift-install/
lab.example.com/hosts-ucf sec-ldap-cleanup.yaml
```

This concludes this demonstration.



## References

Example file with the Ansible inventory file.  
<https://github.com/openshift/openshift-ansible/blob/master/inventory/hosts.example>

Further information is available in the *Configuring Authentication* section from the chapter of the *Administration Guide* for OpenShift Container Platform 3.6; at  
<https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Configuring the LDAP Security Provider

In this exercise, you will configure LDAP as the security provider for OpenShift Container Platform.

## Outcomes

You should be able to customize an existing OpenShift cluster installation to authenticate using LDAP.

### Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab sec-provider setup** command to download the required files for this exercise and install the LDAP client on the **workstation** VM:

```
[student@workstation ~]$ lab sec-provider setup
```

## Steps

- From **workstation**, log in to the **console** VM as the **root** user:

```
[student@workstation ~]$ ssh root@console
```

- Inspect the file located at **/root/D0380/labs/sec-provider/sec-provider-setup.yaml**.

The Ansible Playbook backs up the master node configuration file and replaces it with the one available at **/root/D0380/labs/sec-provider**.

```

- hosts: masters
 become: yes
 tasks:
 - name: Backup the master-config.yaml file from the host.
 command:
 mv /etc/origin/master/master-config.yaml /etc/origin/master/master-
config-9.2.yaml

 - name: Copy the master-config.yaml file with the LDAP security provider
 copy:
 src: master-config.yaml
 dest: /etc/origin/master/master-config.yaml

 - name: Restart the master API
 systemd:
 state: restarted
 name: atomic-openshift-master-api
```

Close the file.

- Configure the **master-config.yaml** file to use the LDAP provider.

---

From the **console** VM, open the configuration file located at **/root/D0380/labs/sec-provider/master-config.yaml**. Look for the **identityProviders** section:

```
identityProviders:
- challenge: true
 login: true
 mappingMethod: claim
 name: ldap_auth
 provider:
 apiVersion: v1
 kind: #add provider here
 attributes:
 id:
 - dn
 preferredUsername:
 - uid
 insecure: true
 url: "ldap://services.lab.example.com:389/ou=people,dc=redhat,dc=com?uid"
 bindDN: ""
 bindPassword: ""
```

Replace **#add provider here** with **LDAPPasswordIdentityProvider**.

Save and close the file.

4. Test the LDAP search.

On the **workstation** VM, run the OpenLDAP client tool to search for entries:

```
[student@workstation ~]$ ldapsearch -x -H ldap://services.lab.example.com:389/ \
-b "ou=people,dc=redhat,dc=com" -s sub
extended LDIF
#
LDAPv3
base <ou=people,dc=redhat,dc=com> with scope subtree
filter: (objectclass=*)
requesting: ALL
#
people, redhat.com
dn: ou=people,dc=redhat,dc=com
ou: people
description: All people in organisation
objectClass: organizationalUnit

admin, people, redhat.com
dn: uid=admin,ou=people,dc=redhat,dc=com
objectClass: top
objectClass: inetOrgPerson
uid: admin
cn: Joe
sn: Admin
userPassword:: YWRtaW4=
...output omitted...
```

The **ldapsearch** command shows a set of users named **btx** with the password **ldapX**.

5. On the **console** VM, run the Ansible Playbook to update the **master-config.yaml** configuration file:

```
[root@console ~]# cd DO380/labs/sec-provider
[root@console sec-provider]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf sec-provider-setup.yaml
```

6. Ensure that the new user can connect to the OpenShift cluster.

From the **workstation** VM, log in to the cluster using **bt1** as the user name, and **ldap1** as the password:

```
[student@workstation ~]$ oc login -u bt1 -p ldap1 https://console.lab.example.com
Login successful.

You don't have any projects. You can try to create a new project, by running
oc new-project <projectname>
```

7. Add the cluster administration role to the user.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ ssh root@master1 oc adm policy add-cluster-role-to-user \
cluster-admin bt1
cluster role "cluster-admin" added: "bt1"
```

8. Review the roles associated with the user.

On the **workstation** VM, run the following command to list the authentication rules:

```
[student@workstation ~]$ ssh root@master1 oc get users
NAME UID FULL
NAME IDENTITIES
admin 3014d8c4-c8b1-11e7-9ac8-52540000fa0d
 htpasswd_auth:admin
developer 2bed8f45-c8ac-11e7-9ac8-52540000fa0d
 htpasswd_auth:developer
bt1 0fe949ba-c980-11e7-869e-52540000fa0d
 ldap_auth:uid=bt1,ou=people,dc=redhat,dc=com
```

9. Synchronize the LDAP groups.

- 9.1. From **workstation**, open a new terminal and log in to **master1** as the **root** user.

```
[student@workstation ~]$ ssh root@master1
```

- 9.2. Synchronize the LDAP group with the **oc adm groups sync** command. The **/root/sync-ldap-groups.yaml** YAML file contains the configuration for interacting with the LDAP server:

```
...
kind: LDAPSsyncConfig
apiVersion: v1
url: "ldap://services.lab.example.com:389"
insecure: true
```

```
rfc2307:
 groupsQuery:
 baseDN: "cn=customer,ou=Roles,dc=redhat,dc=com"
 scope: sub
 derefAliases: never
 filter: (objectClass=groupOfNames)
```

- 9.3. Run the **oc adm groups sync** command against the configuration file. Ensure that the output lists the users.

```
[root@master1 ~]# oc adm groups sync --sync-config=sync-ldap-groups.yaml
apiVersion: v1
items:
- apiVersion: v1
 kind: Group
 metadata:
 annotations:
 openshift.io/ldap.sync-time: 2017-12-05T12:13:08-0800
 openshift.io/ldap.uid: cn=customer,ou=Roles,dc=redhat,dc=com
 openshift.io/ldap.url: services.lab.example.com:389
 creationTimestamp: 2017-12-05T18:30:39Z
 labels:
 openshift.io/ldap.host: services.lab.example.com
 name: customer
 resourceVersion: "1041484"
 selfLink: /oapi/v1/groups/customer
 uid: 64d6cd30-d9ea-11e7-b244-52540000fa0d
 users:
- bt1
- bt2
- bt3
- bt4
- bt5
 kind: List
 metadata: {}
```

- 9.4. Rerun the previous command with the **--confirm** option to import the groups.

```
[root@master1 ~]# oc adm groups sync --sync-config=sync-ldap-groups.yaml --
confirm
group/customer
```

10. Inspect the groups.

- 10.1. From **master1**, ensure that the groups were imported into the OpenShift cluster database.

```
[root@master1 ~]# oc get groups
NAME USERS
customer bt1, bt2, bt3, bt4, bt5
```

- 10.2. Delete the **customer** group.

```
[root@master1 ~]# oc delete groups customer
group "customer" deleted
```

10.3.Delete the **bt1** user.

```
[root@master1 ~]# oc delete user bt1
user "bt1" deleted
```

11. Revert the configuration.

From the **console** VM, run the following command to revert the authentication settings:

```
[root@console sec-provider]# ansible-playbook \
-i /root/openshift-install/lab.example.com/hosts-ucf sec-provider-cleanup.yaml
```

This concludes the guided exercise.

# Configuring User Security

## Objectives

After completing this section, students should be able to:

- Describe OpenShift Container Platform API authentication.
- Impersonate users and **sudoer** roles.
- Describe OpenShift Container Platform authorizations such as rules, roles, and bindings.
- Discuss user and identity lists.

## Virtual Groups in OpenShift Container Platform

In OpenShift Container Platform, a user is an entity that makes requests to the OpenShift Container Platform API. This is typically the account of a developer or an administrator who is interacting with OpenShift Container Platform. A user can be assigned to one or more groups, each representing a certain set of users. Groups are useful when managing authorization policies to grant permissions to multiple users at once, for example allowing access to objects within a project, versus granting them to users individually.

Virtual groups are automatically provisioned by OpenShift Container Platform. These groups can be retrieved when viewing cluster bindings, as described with the following command:

```
[root@demo ~]# oc describe clusterPolicy default
Name: default
Created: 4 hours ago
Labels: <none>
Last Modified: 2015-06-10 17:22:25 +0000 UTC
admin Verbs Resources
 [create delete get list update watch] [pods/proxy projects ...]
 [get list watch] [pods/exec pods/portforward ...]
 [get update] [imagestreams/layers] ...
...
...
```

In the default set of virtual groups, note the following groups:

- **system:authenticated**: This group is automatically associated with all authenticated users.
- **system:authenticated:oauth**: This group is automatically associated with all authenticated users with an **OAuth** access token.
- **system:unauthenticated**: This group is automatically associated with all unauthenticated users.

## Managing Authorizations in OpenShift Container Platform

Roles apply various levels of access and policies, which includes both cluster and local policies. Users and groups can be associated with multiple roles at the same time. Run the **oc describe** command to view details about the roles and their bindings.

Users with the **cluster-admin** default role in the cluster policy can view cluster policy and all local policies. Users with the **admin** default role in a given local policy can view the policy on a per-project basis.

To view the current set of cluster bindings, which show the users and groups that are bound to various roles, run the following command:

```
[root@demo ~]# oc describe clusterPolicyBindings :default
Name: :default
Created: 7 days ago
Labels: <none>
Annotations: <none>
Last Modified: 2017-11-14 16:46:42 -0800 PST
Policy: <none>
RoleBinding[admin]:
 Role: admin
 Users: <none>
 Groups: <none>
 ServiceAccounts: openshift-infra/template-instance-controller
 Subjects: <none>
...
...
```

### Reading Local Policies

Although the list of local roles and their associated rule sets are not viewable within a local policy, all of the default roles are still applicable and can be added to users or groups, except the **cluster-admin** default role. However, the local bindings are viewable.

To view the current set of local bindings, which shows the users and groups that are bound to various roles, run the following command:

```
[root@demo ~]# oc describe policyBindings :default
Name: :default
Namespace: project-review1
Created: 3 days ago
Labels: <none>
Annotations: <none>
Last Modified: 2017-11-13 11:52:47 -0800 PST
Policy: <none>
RoleBinding[admin]:
 Role: admin
 Users: developer
 Groups: <none>
 ServiceAccounts: <none>
 Subjects: <none>
RoleBinding[system:deployers]:
 Role: system:deployer
 Users: <none>
 Groups: <none>
 ServiceAccounts: deployer
 Subjects: <none>
...
...
```



## Note

By default, in a local policy, only the binding for the **admin** role is immediately listed. However, if other default roles are added to users and groups within a local policy, they are listed as well.

### Managing Role Bindings

Adding, or *binding*, a role to users or groups gives the user or group the relevant access granted by the role. You can add and remove roles to and from users and groups using **oc adm policy** commands.

When managing the user and group roles for a local policy using the following operations, a project may be specified with the **-n** option. If it is not specified, the current project is used.

The following table shows some of the operations for managing local policies:

| Command                                                    | Description                                                |
|------------------------------------------------------------|------------------------------------------------------------|
| <b>oc adm policy who-can verb resource</b>                 | Indicates which users can perform an action on a resource. |
| <b>oc adm policy add-role-to-user role username</b>        | Binds a given role to specified users.                     |
| <b>oc adm policy remove-role-from-user role username</b>   | Removes a given role from specified users.                 |
| <b>oc adm policy remove-user username</b>                  | Removes specified users and all of their roles.            |
| <b>oc adm policy add-role-to-group role groupname</b>      | Binds a given role to specified groups.                    |
| <b>oc adm policy remove-role-from-group role groupname</b> | Removes a given role from specified groups.                |
| <b>oc adm policy remove-group groupname</b>                | Removes specified groups and all of their roles.           |

You can also manage role bindings for the cluster policy using the operations described below. The **-n** option is not used for these operations because the cluster policy does not operate at the namespace level.

The following table shows some of the operations for managing cluster policies:

| Command                                                            | Description                                                                 |
|--------------------------------------------------------------------|-----------------------------------------------------------------------------|
| <b>oc adm policy add-cluster-role-to-user role username</b>        | Binds a given role to specified users for all projects in the cluster.      |
| <b>oc adm policy remove-cluster-role-from-user role username</b>   | Removes a given role from specified users for all projects in the cluster.  |
| <b>oc adm policy add-cluster-role-to-group role groupname</b>      | Binds a given role to specified groups for all projects in the cluster.     |
| <b>oc adm policy remove-cluster-role-from-group role groupname</b> | Removes a given role from specified groups for all projects in the cluster. |

For example, to give the **admin** role to the **developer** user in the **example** project, run the following command:

```
[root@demo ~]# oc adm policy add-role-to-user admin developer -n example
```

Run the **oc describe policyBindings :default -n project** command to review the binding:

```
[root@demo ~]# oc describe policyBindings :default -n example
Name: :default
Created: 5 minutes ago
Labels: <none>
Last Modified: 2015-06-10 22:00:44 +0000 UTC
Policy: <none>
RoleBinding[admins]:
 Role: admin
 Users: [developer]①
 Groups: []
RoleBinding[system:deployers]:
 Role: system:deployer
 Users: [system:serviceaccount:developer:deployer]
 Groups: []
RoleBinding[system:image-builders]:
 Role: system:image-builder
 Users: [system:serviceaccount:developer-project:builder]
 Groups: []
RoleBinding[system:image-pullers]:
 Role: system:image-puller
 Users: []
 Groups: [system:serviceaccounts:developer-project]
```

① The **alice** user has the **admin** role binding.

## API Authentication

API tokens authenticate themselves to the OpenShift Container Platform API whenever a user uses the OpenShift Container Platform command-line interface or the web console. However, when credentials for a regular user are not available, components usually make API calls independently.

Among the components:

- The replication controllers that make API calls to create or delete pods.
- Applications inside containers that make API calls for discovery purposes.
- External applications that make API calls for monitoring or integration purposes.



### Note

Service accounts provide a flexible way to control API access without sharing a user's credentials.

Requests to the OpenShift Container Platform API are authenticated using the following methods:

## OAuth access tokens

OAuth access tokens are obtained from the OpenShift Container Platform OAuth server using the `master/oauth/authorize` and `master/oauth/token` endpoints.

## X.509 client certificates

These certificates require an HTTPS connection to the API sever. They are verified by the API server against a trusted certificate authority bundle. The API server creates and distributes these certificates to the controllers for authentication purposes.

Any request with an invalid access token or an invalid certificate is rejected by the authentication layer with a **401** error. The authentication layer assigns the **system:anonymous** virtual user and the **system:unauthenticated** virtual group to the request if no access token or certificate is presented. This allows the authorization layer to determine which requests, if any, an anonymous user is allowed to make.

Service accounts authenticate to the API using tokens signed by a private RSA key. The authentication layer verifies the signature using a matching public RSA key. To enable service account token generation, update the `serviceAccountConfig` stanza in the `/etc/origin/master/master-config.yaml` configuration file on the master to specify a private key file and a matching public key file, as shown in the following excerpt:

```
serviceAccountConfig:
 ...
 masterCA: ca.crt①
 privateKeyFile: serviceaccounts.private.key②
 publicKeyFiles:
 - serviceaccounts.public.key③
 - ...
```

- ① Defines the chain certificate to validate the API server's serving certificate.
- ② Defines the RSA key file.
- ③ Defines the public RSA key files. If private key files are provided, then the public key component is used. When multiple public key files are specified, a token is accepted if it can be validated by one of the public keys. This allows rotation of the signing key, while still accepting tokens generated by the previous signer.

## Managing Impersonation and Users

A request to the OpenShift Container Platform API may include an **Impersonate-User** header, which indicates that the requester wants to have the request handled as though it comes from the specified user. This can be done on the command line by passing the `--as=username` option.

For example, before **user1** can impersonate **user2**, **user1** must be authenticated. Then, an authorization check occurs to ensure that **user1** is allowed to impersonate **user2**. If **user1** is requesting to impersonate a service account, such as **system:serviceaccount:namespace:name**, OpenShift Container Platform ensures that **user1** can impersonate the service account named *name* in *namespace*. If the authentication fails, the user request fails with an error code of **403**.

By default, project administrators and editors are allowed to impersonate service accounts in their namespace. The **sudoers** role allows users to impersonate the **system:admin** user, which in turn has cluster administrator permissions.

This grants some protection for someone administering the cluster. For example, the **oc delete nodes --all** command is forbidden, but the **oc delete nodes --all --as=system:admin** command is allowed. Use the **oc adm policy add-cluster-role-to-user sudoer username** command to add a user to that group.

### Managing User and Identity Lists

After new users log in to the OpenShift cluster, an account is created for the user according to the identity provider configured on the master nodes. You can manage the access level of each user.

OpenShift Container Platform user configuration is stored in several locations. Regardless of the identity provider, OpenShift Container Platform stores details internally, such as *role-based access control (RBAC)* information and group membership. To remove user information, you must delete this data in addition to the user account itself.

The **user** and **identity** object types contain user data outside the identification provider. To get the current list of users, run the following command:

```
[root@demo ~]# oc get user
NAME UID FULL NAME IDENTITIES
demo 75e4b80c-dbf1-11e5-8dc6-0e81e52cc949 htpasswd_auth:demo
```

To get the current list of identities, run the following command:

```
[root@demo ~]# oc get identity
NAME IDP NAME IDP USER NAME USER NAME USER UID
htpasswd_auth:demo htpasswd_auth demo demo 75e4b80c-
dbf1-11e5-8dc6-0e81e52cc949
```

Note the matching UID between the two object types. If you attempt to change the authentication provider after starting to use the OpenShift cluster, user names that overlap does not authenticate, because of the entries in the identity list, which still points to the old authentication method.

### Managing User and Group Labels

Some of the common tasks involved in managing users are described below:

- To add a label to a user or a group, run the following command:

```
[root@demo ~]# oc label user/username label
```

For example, if the user name is **user1** and the label is **level=accounting**, run the following command to label the user:

```
[root@demo ~]# oc label user/user1 level=accounting
```

- To remove a label from a user or a group, run the following command:

```
[root@demo ~]# oc label user/username label-
```

- To retrieve the labels for a user or a group, run the following command:

```
[root@demo ~]# oc describe user/username
```

### Deleting a User

The following procedure describes how to delete a user:

1. Delete the user.

For example, to delete the **demo** user, run the following command:

```
[root@demo ~]# oc delete user demo
user "demo" deleted
```

2. Delete the user identity.

The identity of the user is related to the identification provider you use. Retrieve the name of the provider from the user record by running the **oc get user** command.

In the following example, the identity provider is **htpasswd\_auth**. The user cannot log in if the command is not executed:

```
[root@demo ~]# oc delete identity htpasswd_auth:demo
identity "htpasswd_auth:demo" deleted
```

After running these steps, a new account is created in OpenShift Container Platform when the user logs in again. If you want to prevent the user from being able to log in again, which could be the case for example if an employee leaves the company, and you want to permanently delete the account, remove the user from your authentication back end (such as **htpasswd**, Kerberos, or others).

For example, if you are using **htpasswd** as the authentication method, delete the entry in the **htpasswd** file configured in OpenShift Container Platform.



### Note

For any external identification management, such as LDAP, or any other *identity management* product, refer to the documentation to remove the user entry.

## Managing Project Templates

The OpenShift Container Platform API server automatically provisions projects based on the templates identified by the **projectRequestTemplate** parameter of the **master-config.yaml** configuration file. If the parameter is not defined, the API server creates a default template that defines a project with the requested name, and assigns the requesting user the **admin** role for the project.

The following procedure describes how to create your custom project template:

1. Create a default project template:

```
[user@demo ~]$ oc adm create-bootstrap-project-template -o yaml > template.yaml
```

2. Use a text editor to modify the **template.yaml** file by adding objects or modifying existing ones.
3. Load the template, in this example, into the **default** project:

```
[user@demo ~]$ oc create -f template.yaml -n default
```

4. Modify the **master-config.yaml** configuration file to reference the loaded template:

```
...
projectConfig:
 projectRequestTemplate: "default/project-request"
...
```

When a project request is submitted, the API substitutes the following parameters into the template:

| Parameter                      | Description                                                      |
|--------------------------------|------------------------------------------------------------------|
| <b>PROJECT_NAME</b>            | This required value defines the name of the project.             |
| <b>PROJECT_DISPLAY_NAME</b>    | This optional value defines the display name of the project.     |
| <b>PROJECT_DESCRIPTION</b>     | This optional value defines the description of the project.      |
| <b>PROJECT_ADMIN_USER</b>      | This required value defines the name of the administrative user. |
| <b>PROJECT_REQUESTING_USER</b> | This required value defines the name of the requesting user.     |

### Note

The access to the API is granted to developers with the **self-provisioner** role and the **self-provisioners** cluster role binding. These roles are available to all authenticated developers by default.

## Managing Node Selectors

You can control pod placement with node selectors, in conjunction with node labels. You can assign labels during the advanced installation, or add them after a node installation.

### Defining Cluster-wide Node Selectors

You can set cluster-wide default node selectors to restrict pod placement to specific nodes. To do so, edit the master configuration file located at **/etc/origin/master/master-config.yaml** on each master node. This is applied to the pods created in all projects without a specified **nodeSelector** value.

The following excerpt shows how to define a node selector:

```
...
projectConfig:
 defaultNodeSelector: "type=user-node,region=east"
...
```

After defining the label, restart the OpenShift Container Platform services:

```
[root@demo ~]# systemctl restart atomic-openshift-master-controller
[root@demo ~]# systemctl restart atomic-openshift-master-api
```

### Defining Project-wide Node Selectors

To define an individual project with a node selector, use the **--node-selector** option when creating a project. For example, if you have an OpenShift Container Platform topology with multiple regions, use a node selector to restrict specific projects to only deploy pods onto nodes in a specific region.

The following command creates a new project called **myproject** and instructs pods to be deployed onto nodes with the **user-node** and **east** labels:

```
[user@demo ~]# oc adm new-project myproject \
--node-selector='type=user-node,region=east'
```

After running the command, all pods deployed in the project are placed according to the selector, as defined by the administrative user.



### Note

Creating a new project with a node selector is only available with the **oc adm command** command.

Use the **oc adm new-project** command to add an annotation section to your project. You can edit a project, and change the **openshift.io/node-selector** property to override the default values:

```
...
metadata:
 annotations:
 openshift.io/node-selector: type=user-node,region=east
...
```

If the **openshift.io/node-selector** property is set to an empty string, for example, **oc adm new-project --node-selector=""**, the project does not have any node selector, even if a selector at the level of the cluster is set as a default. This means that as a cluster administrator, you can set a default selector to restrict developer projects to a subset of nodes, while still enabling infrastructure projects, or other projects to schedule the entire cluster.

### Defining Node Selectors for Developers

If you wish to restrict nodes even further, you can set a node selector on a pod configuration. This is in addition to the project node selector, which means that you can still define node selector values for all projects with a node selector.

For example, if a project is created with the **openshift.io/node-selector: type=user-node, region=east** property, and a developer defines another node selector on a pod in that project, such as **clearance=classified**, the pod is going to be scheduled on nodes that have all three labels: **type=user-node, region=east**, and **clearance=classified**.

If the developer sets the **region=west** selector on a set of pods, the pods would require nodes with the **region=east** and **region=west**, which is not possible. Therefore, the pod is never scheduled, because you can only give one value to a label.



## References

Further information is available in the Configuring Service Accounts section of the *OpenShift Container Platform Cluster Administration Guide* at  
| <https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Managing Authorization Policies section of the *OpenShift Container Platform Cluster Administration Guide* at  
| <https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Configuring User Security

In this exercise, you will impersonate an ordinary user to access administration tools to customize an OpenShift Container Platform container.

## Outcomes

You should be able to customize user permissions to enable access to resources available only to cluster administrators.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab sec-user setup** command to download the required files for this exercise verify that the OpenShift cluster is running:

```
[student@workstation ~]$ lab sec-user setup
```

## Steps

1. Enable privileged access to the **deployer** user.

The **deployer** user is responsible for customizing the environment for developers and must have access to projects created by other users. This includes the template customization that is used in OpenShift Container Platform.

- 1.1. On the **workstation** VM, run the following command to log in to the OpenShift cluster:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

- 1.2. Run the following command to add the **sudoer** role to the **deployer** user:

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user sudoer deployer
cluster role "sudoer" added: "deployer"
```

2. As the **deployer** user, change the permissions on the **sec-user** project.

After executing the previous step, the **deployer** user may change permissions using the **--as** option.

- 2.1. From the **workstation** VM, log in to the cluster as the **deployer** user:

```
[student@workstation ~]$ oc login -u deployer -p redhat \
https://console.lab.example.com
```

- 2.2. Enable access to the **sec-user** project for the **deployer** user.

To ensure that the **deployer** user does not already have permission to access the project run the following command:

```
[student@workstation ~]$ oc project sec-user
error: You are not a member of project "sec-user".
```

```
You are not a member of any projects. You can request a project to be created
with the 'new-project' command.
```

Try to enable access to the **sec-user** project run the following command:

```
[student@workstation ~]$ oc adm policy add-role-to-user admin deployer \
-n sec-user
Error from server (Forbidden): User "deployer" cannot list rolebindings in
project "sec-user"
```

This error occurs because you have not impersonated a user with the correct permissions.

```
[student@workstation ~]$ oc adm policy add-role-to-user admin deployer \
-n sec-user --as=system:admin
role "admin" added: "deployer"
```

Provide the **deployer** user with access to the **sec-user** project.

The **sec-user** project is accessible only by the **developer** user. However, to allow the **deployer** user to customize the environment, you need to provide access to the project as the project administrator.

Run the following command to provide the **deployer** user access to the **sec-user** project:

```
[student@workstation ~]$ oc adm policy add-role-to-user admin deployer \
-n sec-user
role "admin" added: "deployer"
```

### 3. Update the template from OpenShift Container Platform.

According to the environment rules, the MySQL database pod must be deployed to the **apps1** and **apps2** nodes.

#### 3.1. Inspect the labels from the **apps1** and **apps2** nodes.

On the **workstation** VM, use the impersonation capabilities of the **deployer** user to review the labels in the **apps1** node:

```
[student@workstation ~]$ oc describe nodes apps1.lab.example.com \
--as=system:admin | grep -A5 Labels
Labels: beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
 kubernetes.io/hostname=apps1.lab.example.com
 logging-infra-fluentd=true
 region=primary
 zone=local
```

On the **workstation** VM, use the impersonation capabilities of the **deployer** user to retrieve labels in the **apps1** node. Locate the **destination=ca** label.

```
[student@workstation ~]$ oc describe nodes apps2.lab.example.com \
--as=system:admin | grep -A5 Labels
```

```
Labels: beta.kubernetes.io/arch=amd64
 beta.kubernetes.io/os=linux
 destination=ca
 kubernetes.io/hostname=apps2.lab.example.com
 logging-infra-fluentd=true
 region=primary
```

### 3.2. Inspect the template for MySQL pods.

Open the **/home/student/D0380/labs/sec-user/template.yaml** file with a text editor:

```
...output omitted...
spec:
 replicas: 1
 selector:
 name: ${DATABASE_SERVICE_NAME}
 strategy:
 type: Recreate
 template:
 metadata:
 labels:
 name: ${DATABASE_SERVICE_NAME}
 spec:
 nodeSelector:
 destination: ca
 containers:
 - capabilities: {}
 env:
 - name: MYSQL_USER
 valueFrom:
 secretKeyRef:
 key: database-user
 name: ${DATABASE_SERVICE_NAME}
 - name: MYSQL_PASSWORD
 valueFrom:
 secretKeyRef:
 key: database-password
 name: ${DATABASE_SERVICE_NAME}
 - name: MYSQL_ROOT_PASSWORD
 valueFrom:
 secretKeyRef:
 key: database-root-password
 name: ${DATABASE_SERVICE_NAME}
 - name: MYSQL_DATABASE
 valueFrom:
 secretKeyRef:
 key: database-name
 name: ${DATABASE_SERVICE_NAME}
 image: 'registry.lab.example.com:5000/rhscl/mysql-56-rhel7'
```

The **nodeSelector** group was created in the existing template from OpenShift Container Platform. To minimize the number of changes, use the file located at **/home/student/D0380/labs/sec-user/template.yaml** to replace the existing template.

### 3.3. Run the following command to update the template :

```
[student@workstation ~]$ oc replace -f \
/home/student/D0380/labs/sec-user/template.yaml \
-n openshift --as=system:admin
```

```
template "mysql-ephemeral" replaced
```

4. Create the definition using the template.

On the **workstation** VM, run the following command:

```
[student@workstation ~]$ oc process -n openshift mysql-ephemeral \
-p MYSQL_USER=user -p MYSQL_PASSWORD=redhat -p MYSQL_ROOT_PASSWORD=redhat \
-p MYSQL_VERSION=latest --as=system:admin | oc create -n sec-user -f -
secret "mysql" created
service "mysql" created
deploymentconfig "mysql" created
```



### Note

To minimize typing, you can copy and paste the command available at **/home/student/D0380/labs/sec-user/commands.txt**

5. Evaluate the host where the pod is deployed.

To ensure that the node selector is properly used to deploy the MySQL pod, run the following command:

```
[student@workstation ~]$ oc get -n sec-user pods -o wide
NAME READY STATUS RESTARTS AGE IP NODE
mysql-1-5p082 1/1 Running 0 1m 10.130.2.2
apps2.lab.example.com
```

From this output, you can see that the pod was deployed to the **apps2** node, according to the selector defined in the template.

6. Clean up.

- 6.1. Run the following command to delete all the elements created using the template:

```
[student@workstation ~]$ oc process -n openshift mysql-ephemeral \
-p MYSQL_USER=user -p MYSQL_PASSWORD=redhat -p MYSQL_ROOT_PASSWORD=redhat \
-p MYSQL_VERSION=latest --as=system:admin | oc delete -n sec-user -f -
secret "mysql" deleted
service "mysql" deleted
deploymentconfig "mysql" deleted
```



### Note

To minimize typing, you can copy and paste the command available at **/home/student/D0380/labs/sec-user/commands.txt**

- 6.2. Run the following command to delete the objects created during the setup process:

---

```
[student@workstation ~]$ lab sec-user cleanup
```

This concludes the guided exercise.

# Lab: Configuring Security Providers

In this lab, you will impersonate an ordinary user to access administration capabilities to customize OpenShift Container Platform container.

## Outcomes

You should be able to customize user permissions to enable access to resources available only to cluster administrators.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab sec-review setup** command to download the required files for this exercise and to verify that the OpenShift cluster is running:

```
[student@workstation ~]$ lab sec-review setup
```

## Steps

1. Enable privileged access to the **developer** user.

The **developer** user is responsible for customizing the environment for developers and must have access to projects created by other users. This includes the template customization that is used in OpenShift Container Platform.

- 1.1. From the **workstation** VM, log in to the OpenShift cluster as the **admin** user, using **redhat** as the password:
  - 1.2. Give the **developer** user administrative privileges.

2. Test the impersonation capabilities.

Use the **oc label nodes** command to label the nodes without switching users.

- 2.1. From the **workstation** VM, log in to the OpenShift cluster as **developer** using **redhat** as the password:

- 2.2. Add a new label to the **apps1** and **apps2** nodes.

Add the label **location=brazil** to the **apps1** node and **location=us** to the **apps2** node:

3. Grade your work. On the **workstation** VM, run the following command:

```
[student@workstation ~]$ lab sec-review grade
```

This concludes the lab.

# Solution

In this lab, you will impersonate an ordinary user to access administration capabilities to customize OpenShift Container Platform container.

## Outcomes

You should be able to customize user permissions to enable access to resources available only to cluster administrators.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab sec-review setup** command to download the required files for this exercise and to verify that the OpenShift cluster is running:

```
[student@workstation ~]$ lab sec-review setup
```

## Steps

1. Enable privileged access to the **developer** user.

The **developer** user is responsible for customizing the environment for developers and must have access to projects created by other users. This includes the template customization that is used in OpenShift Container Platform.

- 1.1. From the **workstation** VM, log in to the OpenShift cluster as the **admin** user, using **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

- 1.2. Give the **developer** user administrative privileges.

Give the developer administration privileges, run the following command:

```
[student@workstation ~]$ oc adm policy add-cluster-role-to-user sudoer developer
cluster role "sudoer" added: "developer"
```

2. Test the impersonation capabilities.

Use the **oc label nodes** command to label the nodes without switching users.

- 2.1. From the **workstation** VM, log in to the OpenShift cluster as **developer** using **redhat** as the password:

```
[student@workstation ~]$ oc login -u developer -p redhat \
https://console.lab.example.com
```

- 2.2. Add a new label to the **apps1** and **apps2** nodes.

Add the label **location=brazil** to the **apps1** node and **location=us** to the **apps2** node:

```
[student@workstation ~]$ oc label node apps1.lab.example.com \
location=brazil --as=system:admin
node "apps1.lab.example.com" labeled
```

On the **workstation** VM, impersonate the **developer** user and inspect the labels of the **apps1** node:

```
[student@workstation ~]$ oc label node apps2.lab.example.com \
location=us --as=system:admin
node "apps2.lab.example.com" labeled
```

3. Grade your work. On the **workstation** VM, run the following command:

```
[student@workstation ~]$ lab sec-review grade
```

This concludes the lab.

# Summary

In this chapter, you learned:

- You can configure both users and groups in OpenShift Container Platform to enable one by one or group authorization configuration.
- Security providers allow you to manage, in a pluggable fashion, authentication of users to OpenShift Container Platform. This authentication platform is implemented via a built-in OAuth server which acts as a gateway between users and the cluster.
- A request to the OpenShift Container Platform API may include an Impersonate-User header, which indicates that the requester wants to have the request handled as though it comes from the specified user. This can be done on the command line by passing the **--as=username** option.





## CHAPTER 10

# CONFIGURING NETWORKING OPTIONS

| Overview          |                                                                                                                                                                                                                                                                                                 |
|-------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>Goal</b>       | Configure various networking options.                                                                                                                                                                                                                                                           |
| <b>Objectives</b> | <ul style="list-style-type: none"><li>Describe how to integrate a load balancer, use IP failover, and configure various connectivity settings.</li><li>Configure the multitenant SDN provider.</li><li>Describe how OpenShift manages <code>iptables</code> entries.</li></ul>                  |
| <b>Sections</b>   | <ul style="list-style-type: none"><li>Describing Load Balancing, Failover, and Cluster Connectivity Settings (and Quiz)</li><li>Configuring the Multitenant Software-defined Networking Provider (and Guided Exercise)</li><li>Describing <code>iptables</code> Management (and Quiz)</li></ul> |
| <b>Lab</b>        | Configuring Networking Options                                                                                                                                                                                                                                                                  |

# Describing Load Balancing, Failover, and Cluster Connectivity Settings

## Objectives

After completing this section, students should be able to:

- Describe how to integrate a load balancer.
- Implement IP failover features in OpenShift Container Platform.
- Configure various connectivity settings.

## Implementing Router Sharding

*Sharding* refers to the horizontal partitioning of computer resources in order to effectively divide the workload. OpenShift Container Platform uses router sharding to group routers using a common set of labels in the **metadata** attribute. This approach is useful for binding some routes to a router shard, ensuring uniqueness of the route across the shard.

In an OpenShift Container Platform cluster, the route uniqueness allows secure and insecure versions of the same route to exist within a single shard. This implies that routes have a visible life cycle that moves from created to bound to active.



### Note

In a sharded environment, the first route to access the shard obtains the right to occupy it indefinitely, even across restarts.



### Important

For two or more routes that claim the same host name, the resolution order is based on the age of the route and the oldest route wins the claim to that host. In the case of sharded routers, routes are selected based on their labels matching the router's selection criteria. There is no consistent way to determine when labels are added to a route. Therefore, if an older route claiming an existing host name is relabeled to match the router's selection criteria, it replaces the existing route based on the resolution order mentioned above; that is, the oldest route wins.

### Configuring Router Sharding

To implement router sharding, each router that is part of a shard must be labeled. To label a router, scale it to zero and update the deployment configuration to update the environment variables. Otherwise, the shards are not created. The following commands show how to update a router called **router1**:

```
[root@demo ~]# oc adm router router1 --replicas=0
[root@demo ~]# oc set env dc/router1 ROUTE_LABELS="geo=europe"
[root@demo ~]# oc scale dc/router1 --replicas=3
```

In the previous example, the router has a **geo** label with a value of **europe**. In this use case, the router shard is deployed in Europe, as well as all the routes associated with this router. Moreover, the pods associated with these routes are deployed in a European data center.

Labels support various custom expressions, for example:

```
[root@demo ~]# oc set env dc/router1 ROUTE_LABELS="geo in (europe, apac)"
```

In the previous example, the router has a **geo** label with the values **europe** and **apac**. In this use case, the router shard can be deployed in Asia, and all the routes associated with the router.

The **not** operator is also supported:

```
[root@demo ~]# oc set env dc/router1 ROUTE_LABELS="geo != na"
```

### Customizing Router Sharding with Projects

Similarly to the **ROUTE\_LABELS** environment variable, which involves the labeling of a route, you can select routes based on the labels of the route's namespace labels, with the **NAMESPACE\_LABELS** environment variable. For this purpose, the project must be labeled accordingly. To label the **erp** project with a **router=prod** label, use the following command:

```
[root@demo ~]# oc label namespace erp "router=prod"
```

The following example shows how to modify the **router3** router to serve routes whose namespace has the **router=prod** label:

```
[root@demo ~]# oc set env dc/router3 NAMESPACE_LABELS="router=prod"
```

To change the settings, restart the deployment configuration by scaling the number of routers down and then scaling them back up:

```
[root@demo ~]# oc scale dc/router3 --replicas=0 && oc scale dc/router3 --replicas=1
```

## Route-specific IP Whitelists

OpenShift Container Platform implements the concept of *whitelisting*. This feature limits the IP addresses that have access to a route. This is useful if you need to limit the access of some services to applications, such as aggregated logging or the GlusterFS web console.

To use this feature, annotate the **haproxy.router.openshift.io/ip\_whitelist** property in the route definition. The list requires space-separated IP addresses or CIDRs for the approved source addresses. Any requests from IP addresses that are not in the list are dropped:

```
metadata:
 annotations:
 haproxy.router.openshift.io/ip_whitelist: 192.168.1.10
```

In the previous annotation, only the **192.168.1.10** IP address can access the route.

For multiple IP addresses, the list must be separated with spaces:

```
metadata:
 annotations:
```

```
haproxy.router.openshift.io/ip_whitelist: 192.168.1.10 192.168.1.11 192.168.1.12
```

For a CIDR network, the list must use the CIDR notation, as follows:

```
metadata:
 annotations:
 haproxy.router.openshift.io/ip_whitelist: 192.168.1.0/24
```

## Working with Cluster DNS

OpenShift Container Platform has a built-in DNS so that the services are reachable by the service DNS as well as the service IP address and service port. OpenShift Container Platform supports split DNS by running SkyDNS on the master that answers DNS queries for services.

On a node host, each container points to the master node DNS, and the default search domain for the container will be ***pod\_namespace.cluster.local***. The container then directs any DNS queries to the master before any other DNS server on the node, which is the default behavior for containers images. The master answers queries on the ***.cluster.local*** domain, described in the following table:

| <b>DNS Names</b>   |                                                |
|--------------------|------------------------------------------------|
| <b>Object Type</b> | <b>Example</b>                                 |
| Default            | <i>pod_namespace.cluster.local</i>             |
| Services           | <i>service.pod_namespace.svc.cluster.local</i> |
| Endpoints          | <i>name.namespace.svc.cluster.local</i>        |

For container developers, this is a major difference because containers usually require environment variables to send the IP address of other containers needed by an application, such as a database IP address. With the DNS names, the environment variables can be removed without system disruption or system unavailability.



### Note

In a traditional development process, containers use environment variables to get runtime information, such as ports and IP addresses managed by the Docker service. This approach has a major drawback: whenever a service is restarted, all the pods must be restarted, because the new IP address from the service is different from the one in the container's environment variables.

To ensure that the DNS server is running and accessible through a node, check the log messages from the node. Locate the following message:

```
0308 19:51:03.118430 4484 node.go:197] Started Kubelet for node openshiftdev.local,
server at 0.0.0.0:10250
I0308 19:51:03.118459 4484 node.go:199] Kubelet is setting 10.0.2.15 as a DNS nameserver
for domain "local"
```

## Integrating Load Balancers with Plug-ins

OpenShift Container Platform supports hardware load balancers, such as F5 BIG-IP, via router plug-ins. As of OpenShift Container Platform 3.4, with the native integration of F5, OpenShift

Container Platform does not require configuring a node for F5 to be able to reach the pods on the overlay network, as created by OpenShift Container Platform SDN.

The F5 controller pod needs to be launched with the proper configuration so that it can successfully connect directly to pods. To configure it, perform the following procedure:

1. Create a host subnet on the OpenShift Container Platform cluster using the following YAML excerpt:

```
{
 "kind": "HostSubnet",
 "apiVersion": "v1",
 "metadata": {
 "name": "f5-node", ①
 "annotations": {
 "pod.network.openshift.io/assign-subnet": "true",
 "pod.network.openshift.io/fixed-vnid-host": "0" ②
 }
 },
 "host": "openshift-f5-node",
 "hostIP": "10.3.89.213" ③
}
```

- ① Defines the subnet name.
- ② Ensures that F5 encapsulates all packets with a VNID of 0.
- ③ Defines the internal IP address of the F5 appliance.

2. Retrieve the subnet allocated for the host subnet with the following command:

| NAME                  | HOST                  | HOST IP     | SUBNET        |
|-----------------------|-----------------------|-------------|---------------|
| f5-node               | openshift-f5-node     | 10.3.89.213 | 10.131.0.0/23 |
| openshift-master-node | openshift-master-node | 172.17.0.2  | 10.129.0.0/23 |
| openshift-node-1      | openshift-node-1      | 172.17.0.3  | 10.128.0.0/23 |
| openshift-node-2      | openshift-node-2      | 172.17.0.4  | 10.130.0.0/23 |

Based on the previous YAML file, locate the subnet name. In this case, **f5-node**. Locate the value of the **SUBNET** column.

3. Retrieve the pod network's CIDR:

```
[root@demo ~]# oc get clusternetwork
```

The CIDR should be something similar to **10.128.0.0/14**.

4. To construct the gateway address, select any IP address from the host subnet, **10.131.0.X**, and associate it with the mask of the pod network, **14**. The gateway address thus becomes **10.131.0.X/14**.
5. Configure a service account for the F5 router pod as a privileged user:

```
[root@demo ~]# oc adm policy remove-scc-from-user hostnetwork -z router
[root@demo ~]# oc adm policy add-scc-to-user privileged -z router
```

6. Enable the access to the **node** cluster resource for the service account, using with a custom role. For example, the **system:sdn-reader** role:

```
[root@demo ~]# oc adm policy add-cluster-role-to-user system:sdn-reader \
system:serviceaccount:default:router
```

7. Run the following commands as a cluster administrator to deploy the F5 router pod:

```
[root@demo ~]# oc adm router \
--type=f5-router \①
--external-host=10.3.89.213 \②
--external-host-username=admin \③
--external-host-password=mypassword \④
--external-host-http-vserver=time-server \⑤
--external-host-https-vserver=https-ose-vserver \⑥
--external-host-private-key=/path/to/key \⑦
--host-network=false \
--service-account=router \
--external-host-internal-ip=10.3.89.213 \
--external-host-vxlan-gw=10.131.0.5/14⑧
```

- ① Starts the F5 router.
- ② Identifies the management interface or IP address of the host.
- ③ Specifies the user name to connect to the F5 appliance.
- ④ Specifies the password to connect to the F5 appliance.
- ⑤ Specifies the name of the F5 virtual server for HTTP connections.
- ⑥ Specifies the name of the F5 virtual server for HTTPS connections.
- ⑦ Specifies the path to the SSH private key file for the F5 BIG-IP host.
- ⑧ Specifies the VXLAN F5 gateway.

## Configuring Services with External IP and Node Port for HA

By setting an external IP address for a pod service, OpenShift Container Platform sets up the firewall rules that allow traffic incoming to any cluster node and it is routed to one of the internal pods. This is similar to the internal service IP addresses, but the external IP address instructs OpenShift Container Platform to expose the service externally, at the given IP address, and to use it as a Virtual IP address.



### Important

You must assign the IP address to a node interface on one of the nodes in the cluster.

### Configuring External IP for HA

To set up the external IP, define the IP address in the `/etc/origin/master/master-config.yaml`, in the **networkConfig** section:

```
networkConfig:
 externalIPNetworkCIDRs:
 - 192.168.120.0/24
```

To make the changes persistent, restart the master services:

```
[root@demo ~]# systemctl restart atomic-openshift-master-api
[root@demo ~]# systemctl restart atomic-openshift-master-controllers
```

For each service that must be exposed, update the **externalIPs** attribute. To do so, run the following command:

```
[root@demo ~]# oc patch svc name -p '{"spec": {"externalIPs": ["ip_address"]}}'
```

The *ip\_address* defines the IP address that is reserved for the service in an external environment.

On each host that is part of the cluster, enable the IP address exposed by the configuration and configure the network routes.



### Note

For additional information about networking configuration, consult the references.

### Configuring Node Ports for HA

To configure the external IP, define the IP address in the **networkConfig** section of the `/etc/origin/master/master-config.yaml` configuration:

```
networkConfig:
 externalIPNetworkCIDRs:
 - 192.168.120.0/24
```

Restart the master services to make the changes persistent:

```
[root@demo ~]# systemctl restart atomic-openshift-master-api
[root@demo ~]# systemctl restart atomic-openshift-master-controllers
```

Update the **type:NodePort** attribute for each service that you expose by running the following command:

```
[root@demo ~]# oc patch svc name -p '{"spec": {"type": "NodePort"}}'
[root@demo ~]# oc patch svc name -p '{"spec": {"ports": {"nodePort: portNumber"}}}'
```

The *portNumber* defines the port exposed by the service.

As opposed to the configuration of an external IP, this configuration does not require any extra network configuration.

## Implementing High Availability

OpenShift Container Platform uses IP failover pods to implement high availability for pods and services in a cluster. To guarantee that an IP failover pod is running on the nodes, a deployment configuration is deployed to the cluster, which deploys the application pods that require high availability.

The IP fail over pods monitor a port on a *virtual IP address (VIP)*, which is externally visible. This VIP is used to determine if the port is reachable on the node. If the port is reachable, the VIP is assigned to the node. To remove that verification, give the port a value of zero.

### Note

It is currently not possible to explicitly distribute VIPs across nodes.

OpenShift Container Platform supports the creation of IP failover via deployment configurations, using the **oc adm ipfailover** command.

#### Implementing IP Failover

The following procedure describes how to set up a highly available router and geocaching for network services with IP failover on a set of nodes.

1. Label the nodes that are used for the network services. This step is optional if you run the services on all the nodes in your OpenShift cluster and use VIPs that float between all nodes in the cluster.

The following example defines a label for nodes that are servicing traffic in one set of nodes labeled **ha-svc-nodes=ny**:

```
[root@demo ~]# oc label node apps1.lab.example.com "ha-svc-nodes=ny"
```

2. Create the service account.

You can use IP failover with a router. To do so, you can either reuse the router service account created previously, or a new service account for the IP failover feature.

The following example creates a new service account called **ipfailover** in the default namespace:

```
[root@demo ~]# oc create serviceaccount ipfailover -n default
```

3. Add a security context constraint to the **ipfailover** service account in the default namespace:

```
[root@demo ~]# oc adm policy add-scc-to-user privileged
system:serviceaccount:default:ipfailover
```

4. Start the router and the geocaching services:

```
[root@demo ~]# oc adm router ha-router-us-west --replicas=5 \
--selector="ha-svc-nodes=ny" \
```

```
--labels="ha-svc-nodes=ny" \
--service-account=ipfailover
```



## Important

Run the router or service on the nodes labeled in the first step of this procedure, because the IP failover service will be running on these nodes.

5. Deploy the application that requires high availability.

Define the labels that identify the destination nodes and the service from the application, using the **ClusterIP** type:

```
"kind": "Service",
"apiVersion": "v1",
"metadata": {
 "name": "app-service"
},
"spec": {
 "ports": [
 {
 "protocol": "TCP",
 "port": 9736,
 "targetPort": 0,
 "nodePort": 0
 }
],
 "selector": {
 "ha-cache": "geo"
 },
 "type": "ClusterIP",
 "sessionAffinity": "None"
```

6. Configure IP failover for the router and the services. Each service has its own VIP, and both use the same nodes with the **ha-svc-nodes=ny** label. Ensure that the number of replicas matches the number of nodes listed in the label setup, as shown in the first step.
7. Specify the VIPs and the port number that IP failover should monitor on the desired instances:

```
[root@demo ~]# oc adm ipfailover ipf-ha-router-ny \
--replicas=5 --watch-port=80 \
--selector="ha-svc-nodes=ny" \
--virtual-ips="10.245.2.101-105" \
--iptables-chain="INPUT" \
--service-account=ipfailover --create
```

8. Start the IP failover for the service. Because there are two IP failover deployment configurations, set the **--vrrp-id-offset** option so that each VIP gets its own offset. In this case, a value of **10** indicates that the **ipf-ha-router-ny** deployment can have a maximum of ten VIPs, ranging from zero to nine, because the **ipf-ha-app** IP failover pod starts at ten:

```
[root@demo ~]# oc adm ipfailover ipf-ha-app \
--replicas=5 --watch-port=9736 \
--selector="ha-svc-nodes=ny" \
--virtual-ips=10.245.3.101-105 \
--vrrp-id-offset=10 \
--service-account=ipfailover \
--create
```

In the previous command, an IP failover, a router, and an application pod all exist on the same node. The set of VIPs for each IP failover configuration must not overlap, and must not be used elsewhere in any external environment. The five VIP addresses in the previous configuration, **10.245.{2,3}.101-105**, are served by the two IP failover deployment configurations.

## References

Further information is available in the Using the F5 Router Plug-in chapter of the *OpenShift Container Platform Installation and Configuration Guide for OpenShift Container Platform 3.6* at

<https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Create a Project and Service section of the *OpenShift Container Platform Developer Guide for OpenShift Container Platform 3.6* at

<https://access.redhat.com/documentation/en-US/index.html>

# Quiz: Describing Load Balancing, Failover, and Cluster Connectivity Settings

Choose the correct answers to the following questions:

1. Which two statements describe router sharding in OpenShift Container Platform? (Choose two.)
  - a. Each route can be configured to be sharded individually.
  - b. A router can be configured to be sharded.
  - c. The **oc shard routeName** command is used to shard routes.
  - d. To enable sharding, each router must have a set of labels that identifies the shard.
2. Which two statements describe the router IP whitelisting feature? (Choose two.)
  - a. OpenShift Container Platform by default blocks all access to a route. To enable access, whitelist the IP address.
  - b. To configure IP whitelisting, use the **oc route whitelist** command.
  - c. To configure IP whitelisting, use the annotation **haproxy.router.openshift.io/ip\_whitelist** whenever a route needs to have a whitelist.
  - d. To identify each whitelisted IP address, separate them with commas.
  - e. To identify each whitelisted IP address, separate them with spaces.
3. Which statement defines the advantage of using Cluster DNS in the applications running in an OpenShift cluster?
  - a. Application pods configured with Cluster DNS are harder to configure because the IP addresses used by all pods must be passed as environment variables.
  - b. Application pods configured with Cluster DNS require the restarting of pods whenever a service with the IP address is deleted.
  - c. Cluster DNS uses the **service.pod\_namespacesvc.cluster.local** definition as the default DNS name for services.

## Solution

Choose the correct answers to the following questions:

1. Which two statements describe router sharding in OpenShift Container Platform? (Choose two.)
  - a. Each route can be configured to be sharded individually.
  - b. A router can be configured to be sharded.
  - c. The `oc shard routeName` command is used to shard routes.
  - d. To enable sharding, each router must have a set of labels that identifies the shard.
2. Which two statements describe the router IP whitelisting feature? (Choose two.)
  - a. OpenShift Container Platform by default blocks all access to a route. To enable access, whitelist the IP address.
  - b. To configure IP whitelisting, use the `oc route whitelist` command.
  - c. To configure IP whitelisting, use the annotation `haproxy.router.openshift.io/ip_whitelist` whenever a route needs to have a whitelist.
  - d. To identify each whitelisted IP address, separate them with commas.
  - e. To identify each whitelisted IP address, separate them with spaces.
3. Which statement defines the advantage of using Cluster DNS in the applications running in an OpenShift cluster?
  - a. Application pods configured with Cluster DNS are harder to configure because the IP addresses used by all pods must be passed as environment variables.
  - b. Application pods configured with Cluster DNS require the restarting of pods whenever a service with the IP address is deleted.
  - c. Cluster DNS uses the `service.pod_namespacesvc.cluster.local` definition as the default DNS name for services.

# Configuring the Multitenant Software-defined Networking Provider

## Objectives

After completing this section, students should be able to:

- Discuss alternative network providers for OpenShift Container Platform.
- Describe the various multitenant network plug-ins.
- Describe strategies to control external traffic.
- Install and configure the flannel plug-in.
- Troubleshoot software-defined networking (SDN) in OpenShift Container Platform.

## Introduction to OpenShift Container Platform SDN

The OpenShift Container Platform SDN enables communication between pods across the cluster. Three SDN plug-ins are currently available for OpenShift Container Platform: **ovs-subnet**, **ovs-multitenant**, and **ovs-networkpolicy**.

When using the advanced installation method, the OpenShift Container Platform installer uses the **ovs-subnet** plug-in by default. You can use the **os\_sdn\_network\_plugin\_name** parameter in the Ansible inventory file to specify which plug-in to use. The following excerpt shows how to use the **ovs-multitenant** plug-in as the network plug-in:

```
Configure the multi-tenant SDN plugin
os_sdn_network_plugin_name='redhat/openshift-ovs-multitenant'①

Disable the OpenShift SDN plugin
openshift_use_openshift_sdn=False②

osm_cluster_network_cidr=10.1.0.0/16③
```

- ① Defines **ovs-multitenant** as the SDN plug-in.
- ② Disables the default OpenShift Container Platform SDN plug-in. This is mandatory when using alternative network plug-ins.
- ③ Defines the cluster network CIDR block. This network block should be a private block and should not conflict with existing network blocks in your infrastructure that pods may require access to.



### Important

The CIDR value cannot be changed after deployment.

You can control the pod network settings on the master nodes by updating the parameters in the **networkConfig** section of the **/etc/origin/master/master-config.yaml** master configuration file.

The following excerpt shows how to configure the **networkConfig** section of the master nodes:

```
networkConfig:
 clusterNetworkCIDR: 10.128.0.0/14①
 hostSubnetLength: 9②
 networkPluginName: "redhat/openshift-ovs-subnet"③
 serviceNetworkCIDR: 172.30.0.0/16④
```

- ① Defines the cluster network for node IP addresses allocation.
- ② Specifies the number of bits per node for allocating IP addresses to pods.
- ③ Specifies the network plug-in to use, for example, **ovs-subnet**.
- ④ Specifies the service IP address allocation for the cluster.



## Important

The **serviceNetworkCIDR** and **hostSubnetLength** parameters cannot be changed after the cluster has been created, and the **clusterNetworkCIDR** parameter can only be changed to be a larger network that still contains the original network. For example, given the default value of **10.128.0.0/14**, you could change **clusterNetworkCIDR** to **10.128.0.0/9**, that is, the entire upper half of the network allocation. However, you cannot change to a lower range, for example, **10.64.0.0/16**, which does not overlap the original value.

### Configuring the Pod Network on Nodes

You can control the pod network settings on the cluster nodes by modifying the settings in the **networkConfig** section of the node configuration file, located at **/etc/origin/node/node-config.yaml** in a node. You can update both the *maximum transmission unit (MTU)* and the network plug-in to use.

The following excerpt shows how to configure the pod network settings in the nodes configuration file:

```
networkConfig:
 mtu: 1450①
 networkPluginName: "redhat/openshift-ovs-subnet"②
```

- ① Specifies the MTU for the pod overlay network.



## Important

To prevent packets from being fragmented, the value should not exceed 1450 bytes for VXLAN addressing, because a VXLAN header uses 8 bytes on top of extra headers.

- ② Specifies the network plug-in to use, in this case, **ovs-subnet**.

To change from one network plug-in to another, follow these steps:

1. Update the **networkPluginName** parameter in all master and node configuration files.

2. Restart the **atomic-openshift-master** services on the master node:

```
[root@demo-master ~]# systemctl restart atomic-openshift-master-api
[root@demo-master ~]# systemctl restart atomic-openshift-master-controllers
```

3. Restart the **atomic-openshift-node** service on the nodes:

```
[root@demo-node ~]# systemctl restart atomic-openshift-node
```

4. If you are changing from an OpenShift Container Platform SDN plug-in to a third-party plug-in, run the following commands to clean up any OpenShift Container Platform SDN-specific artifacts, which includes networks, subnets, and network namespaces:

```
[root@demo ~]# oc delete clusternetwork --all
[root@demo ~]# oc delete hostsubnets --all
[root@demo ~]# oc delete namespaces --all
```

### Controlling Egress Traffic

To control egress traffic, you can allocate a number of static IP addresses to a specific node at the host level. If developers need a dedicated IP address for an application service, they can request one during the process used to ask for firewall access. They can then deploy an egress router from a project, using a node selector in the deployment configuration to ensure that the pod lands on the host with the set of preallocated static IP addresses.

The egress pod's deployment declares one of the source IP addresses, the destination IP address of the protected service, and a gateway IP address to reach the destination. After the pod is deployed, you can create a service to access the egress router pod, and then add that source IP address to the external firewall. The developer then has access to the egress router service that was created in their project, for example, **service.project.cluster.lab.example.com**.

You can control egress traffic in two ways:

#### Firewall

You can use an egress firewall to enforce the acceptable outbound traffic policies, so that specific endpoints or IP ranges, or subnets are the only acceptable targets for the dynamic endpoints, that is, pods within OpenShift Container Platform to talk to.

#### Router

You can use an egress router to create identifiable services to send traffic to specific destinations, ensuring external destinations manage traffic as though it were coming from a known source. This is recommended to increase security, because it secures an external service, such as a database, so that only specific pods in a namespace can talk to a service, such as an egress router, which proxies the traffic to application service, for example, a database.

#### Installing Flannel

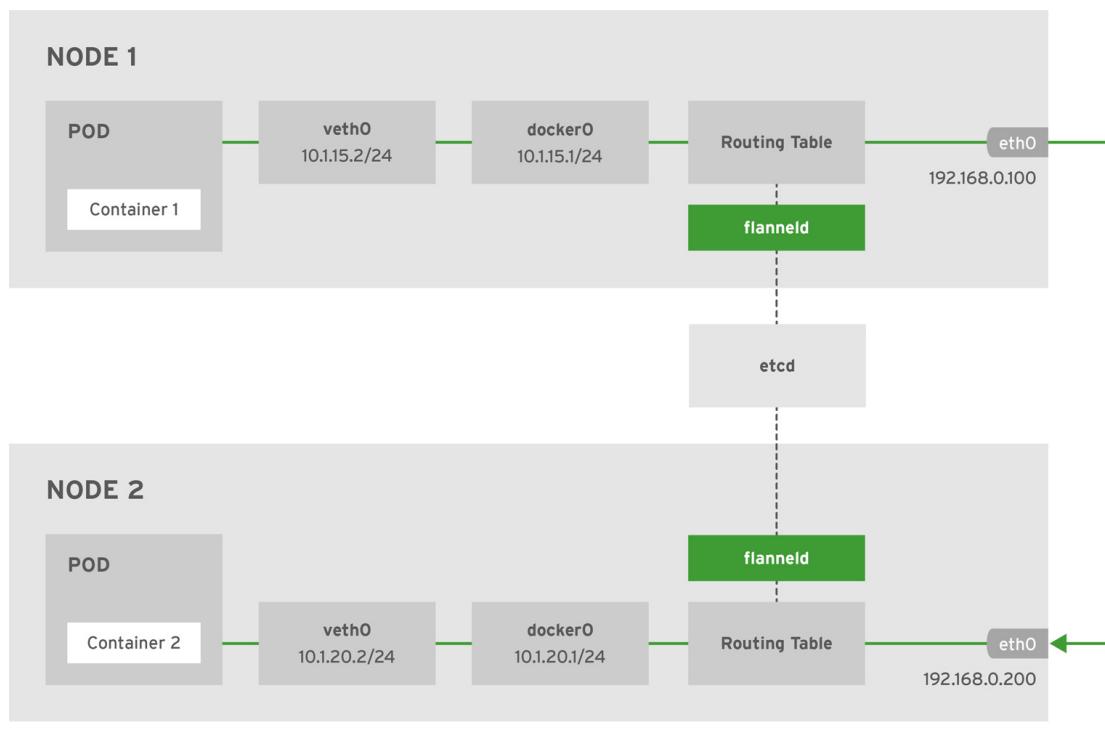
Flannel is a virtual networking layer designed specifically for containers. OpenShift Container Platform can use flannel to network containers instead of using the default SDN components. This is useful if OpenShift Container Platform is deployed within a cloud provider platform that also relies on SDN, such as Red Hat OpenStack Platform, and you want to avoid encapsulating packets twice through both platforms. OpenShift Container Platform provides Ansible Playbooks for installing flannel-based networking.

OpenShift Container Platform runs flannel in **host-gw** mode, which maps routes from container to container. Each host within the network runs an agent called **flanneld**, which is responsible for:

- Managing a unique subnet on each host.
- Distributing IP addresses to each container on its host.
- Mapping routes from one container to another, even if they are running on different hosts.

Each **flanneld** agent provides such information to a centralized Etcd data store so that other agents on hosts can route packets to other containers within the flannel network.

The following diagram illustrates the architecture and data flow from one container to another using a flannel-based network:



*Figure 10.1: Flannel-based network in OpenShift Container Platform*



## Important

The flannel network plug-in is only supported for OpenShift Container Platform deployed on Red Hat OpenStack Platform.



## Important

You must configure Neutron port security to support flannel, even if you do not use security groups.

The following procedure describes how to install and configure the flannel plug-in.

1. In Red Hat OpenStack Platform, configure the Neutron port security controls. The default configuration of Red Hat OpenStack Platform disables user control of port security. Configure Neutron to allow users to control the port security settings on individual ports.

On the Neutron servers, add the following entries to the `/etc/neutron/plugins/ml2/ml2_conf.ini` configuration file:

```
[ml2]
...
extension_drivers = port_security
```

2. Restart the Neutron services:

```
[root@demo ~]# service neutron-dhcp-agent restart
[root@demo ~]# service neutron-ovs-cleanup restart
[root@demo ~]# service neutron-metadata-agent restart
[root@demo ~]# service neutron-l3-agent restart
[root@demo ~]# service neutron-plugin-openvswitch-agent restart
[root@demo ~]# service neutron-vpn-agent restart
[root@demo ~]# service neutron-server restart
```

3. Set the following variables in your Ansible inventory file before running the installation:

```
openshift_use_openshift_sdn=false
openshift_use_flannel=true
flannel_interface=eth0
```

4. Optionally, you can specify the interface to use for interhost communication using the **flannel\_interface** variable. Without this variable, the OpenShift Container Platform installation uses the default network interface.

#### Troubleshooting the OpenShift Container Platform SDN

Multiple interface layers are created to correctly pass traffic between containers. Consequently, you need to test the different stack layers to determine where any connectivity issues might exist.

If you are connected to a server outside the cluster and are trying to access a resource provided by the cluster, you need a process running in a pod that listens on a public IP address and routes that traffic inside the cluster. The OpenShift Container Platform router serves that purpose for HTTP, HTTPS, web sockets, and TLS with *server name indication (SNI)*.

- If you cannot access an HTTP service from outside of the cluster, start by reproducing the problem from the command line on the machine where the service is not accessible:

```
[user@demo ~]$ curl -kv http://service.example.com:8000/application
```

- If the service is reachable, that may indicate that the service has some pods that work, and some that do not. If the service is not reachable, try to resolve the DNS name to an IP address:

```
[user@demo ~]$ dig +short service.example.com
```

- Use the **ping -c address** and **tracepath address** commands to ensure that you can reach the router host. It is possible that the address does not respond to ICMP packets, in which case the tests fail, but the router may be reachable. In this case, try using the **telnet** command to access the port for the router directly:

```
[user@demo ~]$ telnet IP address port
```

If the command fails, that indicates that the router is not listening on the port. At this point, review the routing and firewall configuration.

### Troubleshooting Routers

To troubleshoot routers in OpenShift Container Platform, use the **ssh** command to log in to the server that runs the router and verify that the router software is running and configured correctly.

1. Log in to the cluster and ensure that the router is running, and retrieve the endpoints:

```
[root@demo ~]# oc login -u system:admin -n default
[root@demo ~]# oc get endpoints --namespace=default --selector=router
NAMESPACE NAME ENDPOINTS
default router 10.128.0.4:80
```

If the command fails, it indicates that the OpenShift Container Platform configuration is incorrect. While you should see one or more router endpoints listed, this does not indicate whether the routers are running on the machine with the given external IP address, because the endpoint IP address is one of the pod addresses that is internal to the cluster.

2. To get the list of router host IP addresses, run the following command:

```
[root@demo ~]# oc get pods --all-namespaces --selector=router \
--template='{{range .items}} HostIP: {{.status.hostIP}} \
PodIP: {{.status.podIP}} \
{{end}} {{"\n"}}'
HostIP: 192.168.122.202 PodIP: 10.128.0.4
```

If you do not see the host IP address that matches the external address, configure the router pod to run on the correct node, or update your DNS to match the IP addresses where the routers are running.

3. Ensure that the router is mapping the external URL to the correct service. If that is not the case, you need to review the service configuration to make sure that all endpoints are reachable. Run the following command to list all of the routes in OpenShift Container Platform:

```
[root@demo ~]# oc get route --all-namespaces
NAME HOST/PORT PATH SERVICE
my-route www.example.com /test my-service
```

If the host name and path from the URL do not match anything in the list of returned routes, you need to add a route. If the route is present, you need to debug access to the endpoints.



## Note

For detailed steps on how to troubleshoot a service, node to node networking, virtual networking, and pod egress traffic, consult the Troubleshooting OpenShift SDN section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Cluster Administration Guide* listed in the references.



## References

### Managing Networking

<https://docs.openshift.com/container-platform/3.6/>

Further information is available in the Configuring the SDN section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Installation and Configuration Guide* at  
<https://access.redhat.com/documentation/en-US/index.html>

Further information is available in the Troubleshooting OpenShift SDN section of the *Red Hat Enterprise Linux 7 OpenShift Container Platform Cluster Administration Guide* at  
<https://access.redhat.com/documentation/en-US/index.html>

# Guided Exercise: Configuring the Multitenant Software-defined Networking Provider

In this exercise, you will update an Ansible inventory file to enable support for a multitenant network provider.

## Outcomes

You should be able to:

- Manage a repository using the **git** command.
- Update an Ansible inventory file.

## Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab sdn-configure setup** command. The script installs the required files for this exercise and creates a persistent volume:

```
[student@workstation ~]$ lab sdn-configure setup
```

## Steps

1. On the **workstation** VM, open a new terminal and use the **ssh** command to log in to the **console** VM:

```
[student@workstation ~]$ ssh root@console
```

2. Go to the **openshift-install** repository and retrieve the latest changes by running the **git pull** command:

```
[root@console ~]$ cd openshift-install
[root@console openshift-install]$ git pull
Already up-to-date.
```

3. Go to the **lab.example.com** directory, which contains the Ansible inventory file used to install the OpenShift cluster.

Use a text editor to update the file. Append the following variables to the **[OSEv3:vars]** group.



## Note

Optionally, copy and paste the content of the file located at **/root/D0380/labs/sdn-configure/sdn-variables-inventory.yaml**

- **openshift\_use\_openshift\_sdn=false**: Disables the default OpenShift Container Platform SDN.
- **openshift\_use\_flannel=true**: Enables flannel as the multitenant SDN provider.

- 
- **flannel\_interface=eth0**: Specifies which network interface to use for interhost communication.

The group should read as follows:

```
[OSEv3:vars]
...
openshift_set_node_ip=True
ansible_ssh_user=root

openshift_disable_check=disk_availability,docker_storage, \
memory_availability,docker_image_availability
openshift_cockpit_deployer_prefix='openshift3/'

#running offline
openshift_docker_additional_registries=registry.lab.example.com:5000
openshift_docker_insecure_registries=registry.lab.example.com:5000
openshift_docker_blocked_registries=registry.access.redhat.com,docker.io

openshift_use_openshift_sdn=false
openshift_use_flannel=true
flannel_interface=eth0
```

4. Add the file to the Git index, and commit your changes with a message of **Enables Flannel SDN**.

```
[root@console lab.example.com]# git add .
[root@console lab.example.com]$ git commit -m "Enables Flannel SDN"
[master 2350286] Enables Flannel SDN
...
1 file changed, 29 insertions(+)
```

5. Push your changes to the remote Git server:

```
[root@console lab.example.com]# git push
...
Compressing objects: 100% (4/4), done.
Writing objects: 100% (4/4), 928 bytes | 0 bytes/s, done.
Total 4 (delta 1), reused 0 (delta 0)
To http://services.lab.example.com/openshift-install
 5c13417..2350286 master -> master
```

6. On the **workstation** VM, open a new terminal tab and run the **lab sdn-configure grade** command to ensure that the variables are properly set. The output should read as follows:

```
[student@workstation ~]$ lab sdn-configure grade
Reading variables
 • Checking openshift_use_openshift_sdn=false..... PASS
 • Checking openshift_use_flannel=true..... PASS
 • Checking flannel_interface=eth0..... PASS
```

This concludes the guided exercise.

# Describing iptables Management

## Objective

After completing this section, students should be able to describe the management of IP tables entries.

## iptables Overview

OpenShift Container Platform depends on the Netfilter service to manage network traffic for all services inside and outside the SDN.



### Important

To avoid problems with firewall rules, some components, such as Docker services and OpenShift cluster services, can fix minor inconsistencies, but not larger ones.

If you need to use Netfilter, keep a limited configuration in the Netfilter and rely on OpenShift Container Platform and Docker to install the rules they require.



### Important

If you need to restart the **iptables** service, restart the **docker** and **atomic-openshift-node.service** services to rebuild the firewall rules.

### Reading Netfilter Internal Rules

OpenShift Container Platform configures firewall rules as nonpersistent. This means that the rules are dropped if the firewall service is restarted. Internally, OpenShift Container Platform add rules to a chain called **OS\_FIREWALL\_ALLOW** every time it needs a firewall rule. If you need to customize the rules, they must be manually added on each node using one of following three approaches:

- Update the firewall rules from the command line.

Run the following command as the **root** user:

```
[root@demo ~]# iptables -A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 9000 -j ACCEPT
```

- Update the firewall rules in the **iptables** configuration file.

Update the **/etc/sysconfig/iptables** file on each node in a cluster. This is done to ensure persistence of the rules added via the command line. For each rule, add a line similar to the example presented below, which is based on the **iptables** command above.

```
-A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 9000 -j ACCEPT
```

- Configure the firewall during the installation process.

As part of the installation process in OpenShift Container Platform, use the following Ansible task:

```
- name: Open firewall ports
hosts: all
become: yes
vars:
 os_firewall_allow:
 - service: My monitoring agent at port 9091
 port: 9091/tcp
roles:
 - { role: '/usr/share/ansible/openshift-ansible/playbooks/byo/roles/os_firewall' }
```

To execute the task, run the following command:

```
[root@demo ~]# ansible-playbook /root/my_openshift_playbook_wrapper.yml
```



### Note

Red Hat recommends that you use the Ansible Playbook to eliminate the need to run the **iptables** command on all nodes.

To list the current set of rules used by OpenShift Container Platform, run the following command as the **root** user:

```
[root@demo ~]# iptables -L OS_FIREWALL_ALLOW -n
```

## Limiting Access to External Resources Using Netfilter Rules

Some cluster administrators may want to perform actions on outgoing traffic. You can use Netfilter rules to perform such actions. For example, you can create rules that log traffic to particular destinations, or throttle the number of outgoing connections per second.

OpenShift Container Platform does not provide a way to add custom Netfilter rules automatically, but it does provide a location where you can manually add these rules. When each node starts, it creates an empty chain called **OPENSHIFT-ADMIN-OUTPUT-RULES** in the filter table. Any rule added to the chain applies only to all traffic going from a pod to a destination outside the cluster.

When using Netfilter rules functionality, there are some precautionary steps:

- Ensure that you create the required rules on each node. OpenShift Container Platform does not do that automatically.
- The handling of connections between pods and nodes, or pods and the master is complex, because nodes have both external IP addresses and internal SDN IP addresses. Therefore, some traffic from pods to nodes, and network traffic between masters may pass through the Netfilter chain, but any other pod to node traffic, or master to master traffic may bypass it.

### Enabling Multicasting

In some conditions, network multicasting is required to guarantee that some applications work as expected. OpenShift Container Platform can enable multicasting on a project basis, which avoids a system-wide configuration, which may cause network security breaches. To enable multicasting, run the following command:

```
[root@demo ~]# oc annotate netnamespace namespace \
netnamespace.network.openshift.io/multicast-enabled=true
```



## References

What is the correct method for adding a persistent rule to iptables on an OpenShift Master or Node?

<https://access.redhat.com/solutions/2695741/>

Further information is available in the iptables chapter of the *Cluster Administration Guide for OpenShift Container Platform 3.6* at

<https://access.redhat.com/documentation/en-US/index.html>

## Quiz: Describing OpenShift Container Platform's iptables Management

Choose the correct answers to the following questions:

1. In the following scenario, external systems need to access the LDAP server that is running on one of the nodes, using TCP port 389.

Which command enables access without disrupting the OpenShift Container Platform network configuration?

- a. `iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 389 -j ACCEPT`
- b. `iptables -A OUTPUT -p tcp -m state --state NEW -m tcp --dport 389 -j ACCEPT`
- c. `iptables -A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 389 -j ACCEPT`
- d. `iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 389 -j ACCEPT`

2. In the following scenario, external systems need to access the MySQL database server running on one of the nodes, using TCP port 3306.

Which entry in the `/etc/sysconfig/iptables` configuration file enables access without disrupting the OpenShift Container Platform network configuration?

- a. `-A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 3306 -j ACCEPT`
- b. `-A INPUT -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT`
- c. `-A OUTPUT -p udp -m state --state NEW -m udp --dport 3306 -j ACCEPT`
- d. `-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 3306 -j DROP`
- e. `-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT`

3. Which command enables multicasting for the `odyssey` project?

- a. `oc annotate netnamespace odyssey netnamespace.network.openshift.io/multicast-enabled=true`
- b. `oc adm multicast odyssey`
- c. `oc annotate project odyssey netnamespace.network.openshift.io/multicast-enabled=true`
- d. `oc multicast odyssey`

## Solution

Choose the correct answers to the following questions:

1. In the following scenario, external systems need to access the LDAP server that is running on one of the nodes, using TCP port 389.

Which command enables access without disrupting the OpenShift Container Platform network configuration?

- a. `iptables -A INPUT -p tcp -m state --state NEW -m tcp --dport 389 -j ACCEPT`
- b. `iptables -A OUTPUT -p tcp -m state --state NEW -m tcp --dport 389 -j ACCEPT`
- c. `iptables -A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 389 -j ACCEPT`
- d. `iptables -A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 389 -j ACCEPT`

2. In the following scenario, external systems need to access the MySQL database server running on one of the nodes, using TCP port 3306.

Which entry in the `/etc/sysconfig/iptables` configuration file enables access without disrupting the OpenShift Container Platform network configuration?

- a. `-A OS_FIREWALL_ALLOW -p udp -m state --state NEW -m udp --dport 3306 -j ACCEPT`
- b. `-A INPUT -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT`
- c. `-A OUTPUT -p udp -m state --state NEW -m udp --dport 3306 -j ACCEPT`
- d. `-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 3306 -j DROP`
- e. `-A OS_FIREWALL_ALLOW -p tcp -m state --state NEW -m tcp --dport 3306 -j ACCEPT`

3. Which command enables multicasting for the `odyssey` project?

- a. `oc annotate netnamespace odyssey netnamespace.network.openshift.io/multicast-enabled=true`
- b. `oc adm multicast odyssey`
- c. `oc annotate project odyssey netnamespace.network.openshift.io/multicast-enabled=true`
- d. `oc multicast odyssey`

# Lab: Configuring Networking Options

In this lab, you will troubleshoot the problems created by a network administrator in an OpenShift cluster.

## Outcomes

You should be able to resolve those problems in an OpenShift cluster caused by firewall changes and make the necessary changes.

### Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab network-review setup** command to download the required files for this exercise and verify that OpenShift cluster is running:

```
[student@workstation ~]$ lab network-review setup
```

## Steps

1. Evaluate the changes made to the network.

A new network administrator decided to configure the network for all nodes from OpenShift Container Platform. However, that change caused issues for developers trying to deploy new applications. The developers are reporting that the master API host is not responding to the OpenShift Container Platform client.

- 1.1. Inspect the master API access issue.
- 1.2. Access the master API host to check the settings.
- 1.3. Inspect the services started on the **console** VM.
- 1.4. Verify that the network ports used by the OpenShift master API to enable external access are available.

The port is listening to requests and it is bound to the 0.0.0.0 IP address. Technically, the service is running.

- 1.5. Inspect the firewall rule to determine if external access is blocked.
- 1.6. Access the master API from the **console** VM to inspect the change.
2. Inspect the master node configuration.

The error indicates that the master node could not be reached.

- 2.1. Review the status of the firewall on each master node.

Because OpenShift Container Platform depends on iptables, some rules may not work

- 2.2. To ensure that the master nodes can manage deployments, start the iptables service:

- 
- 2.3. Log in from the **workstation** VM to the master API as **admin** using **redhat** as the password:
3. Grade your work.

This concludes the lab.

## Solution

In this lab, you will troubleshoot the problems created by a network administrator in an OpenShift cluster.

### Outcomes

You should be able to resolve those problems in an OpenShift cluster caused by firewall changes and make the necessary changes.

#### Before you begin

Log in to the **workstation** VM as **student** using **student** as the password.

Run the **lab network-review setup** command to download the required files for this exercise and verify that OpenShift cluster is running:

```
[student@workstation ~]$ lab network-review setup
```

#### Steps

- Evaluate the changes made to the network.

A new network administrator decided to configure the network for all nodes from OpenShift Container Platform. However, that change caused issues for developers trying to deploy new applications. The developers are reporting that the master API host is not responding to the OpenShift Container Platform client.

- 1.1. Inspect the master API access issue.

From the **workstation** VM, log in to the OpenShift cluster as the **admin** user, using **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

The access freezes and no output is provided.

- 1.2. Access the master API host to check the settings.

Log in to the **console** VM as **root** using **redhat** as the password:

```
[student@workstation ~]$ ssh root@console
```

- 1.3. Inspect the services started on the **console** VM.

Use the **systemctl status** command to determine if the OpenShift HA proxy is accessible.

```
[root@console ~]$ systemctl status haproxy
● haproxy.service - HAProxy Load Balancer
 Loaded: loaded (/usr/lib/systemd/system/haproxy.service; enabled; vendor
 preset: disabled)
 Drop-In: /etc/systemd/system/haproxy.service.d
 └─limits.conf
 Active: active (running) since Wed 2017-11-15 19:19:47 EST; 29min ago
 Main PID: 18089 (haproxy-systemd)
```

```

CGroup: /system.slice/haproxy.service
├─18089 /usr/sbin/haproxy-systemd-wrapper -f /etc/haproxy/haproxy.cfg
-p /run/haproxy.pid
└─18090 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/
haproxy.pid -Ds
└─18091 /usr/sbin/haproxy -f /etc/haproxy/haproxy.cfg -p /run/
haproxy.pid -Ds

Nov 15 19:19:47 console.lab.example.com systemd[1]: Started HAProxy Load
Balancer.
Nov 15 19:19:47 console.lab.example.com systemd[1]: Starting HAProxy Load
Balancer...
Nov 15 19:19:47 console.lab.example.com haproxy[18090]: Proxy stats started.
Nov 15 19:19:47 console.lab.example.com haproxy-systemd-wrapper[18089]: haproxy-
systemd-wrapper: executing /usr/sbin/haproxy -f /etc/haproxy/ha...d -Ds
Nov 15 19:19:47 console.lab.example.com haproxy-systemd-wrapper[18089]:
[WARNING] 318/191947 (18090) : config : 'option forwardfor' ignored for...mode.
Nov 15 19:19:47 console.lab.example.com haproxy-systemd-wrapper[18089]:
[WARNING] 318/191947 (18090) : config : 'option forwardfor' ignored for...mode.
Nov 15 19:19:47 console.lab.example.com haproxy[18090]: Proxy atomic-openshift-
api started.
Nov 15 19:19:47 console.lab.example.com haproxy[18090]: Proxy atomic-openshift-
api started.
Hint: Some lines were ellipsized, use -l to show in full.

```

The **haproxy** service is started and running,, which means that the issue lies elsewhere.

- Verify that the network ports used by the OpenShift master API to enable external access are available.

Use the **netstat** command to evaluate the port the OpenShift master API is using:

```
[root@console ~]# netstat -tnlp | grep 443
tcp 0 0 0.0.0.0:443 0.0.0.0:* LISTEN
18091/haproxy
```

The port is listening to requests and it is bound to the 0.0.0.0 IP address. Technically, the service is running.

- Inspect the firewall rule to determine if external access is blocked.

To review the rules, run the following command:

```
[root@console ~]# iptables -L
Chain INPUT (policy ACCEPT)
target prot opt source destination
DROP tcp -- anywhere anywhere tcp dpt:https
...
```

Port 443 is blocked by the **iptables** service. To remove this limitation, restart the service:

```
[root@console ~]# systemctl restart iptables
```

- Access the master API from the **console** VM to inspect the change.

From the **workstation** VM, log in to the OpenShift cluster as the **admin** user, using **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
error: EOF
```

The error is different and it does not allow you to run the commands.

## 2. Inspect the master node configuration.

The error indicates that the master node could not be reached.

### 2.1. Review the status of the firewall on each master node.

On the **console** VM, run the following commands:

```
[root@console ~]# ssh root@master1 systemctl status iptables
● iptables.service - IPv4 firewall with iptables
 Loaded: loaded (/usr/lib/systemd/system/iptables.service; disabled; vendor
 preset: disabled)
 Active: inactive (dead) since Thu 2017-11-16 05:33:49 EST; 4min 9s ago
 Main PID: 731 (code=exited, status=0/SUCCESS)
```

```
[root@console ~]# ssh root@master2 systemctl status iptables
● iptables.service - IPv4 firewall with iptables
 Loaded: loaded (/usr/lib/systemd/system/iptables.service; disabled; vendor
 preset: disabled)
 Active: inactive (dead) since Thu 2017-11-16 05:33:49 EST; 4min 9s ago
 Main PID: 731 (code=exited, status=0/SUCCESS)
```

```
[root@console ~]# ssh root@master3 systemctl status iptables
● iptables.service - IPv4 firewall with iptables
 Loaded: loaded (/usr/lib/systemd/system/iptables.service; disabled; vendor
 preset: disabled)
 Active: inactive (dead) since Thu 2017-11-16 05:33:49 EST; 4min 9s ago
 Main PID: 731 (code=exited, status=0/SUCCESS)
```

Because OpenShift Container Platform depends on iptables, some rules may not work

### 2.2. To ensure that the master nodes can manage deployments, start the iptables service:

```
[root@console ~]# ssh root@master1 systemctl start iptables
```

```
[root@console ~]# ssh root@master2 systemctl start iptables
```

```
[root@console ~]# ssh root@master3 systemctl start iptables
```

### 2.3. Log in from the **workstation** VM to the master API as **admin** using **redhat** as the password:

```
[student@workstation ~]$ oc login -u admin -p redhat \
https://console.lab.example.com
```

3. Grade your work.

From the **workstation** VM run the following command to grade your work:

```
[student@workstation ~]$ lab network-review grade
```

This concludes the lab.

## Summary

In this chapter, you learned:

- OpenShift Container Platform supports route sharding, which allows administrators to distribute the routers to a specific set of nodes.
- To enable route sharding, change the environment variable using the **oc set env dc/router1 ROUTE\_LABELS="labels"**.
- To enable access to a certain list of IP addresses, OpenShift Container supports the whitelisting concept. To enable whitelisting, annotate the router with **haproxy.router.openshift.io/ip\_whitelist**. Separate each IP address with a space.
- OpenShift Container Platform supports Cluster DNS which provides an internal address for service, node, and pod.
- To enable external load balancer support, such as F5 BIG-IP:
  - Create a **HostSubNet** object that points to the F5 virtual IP address.
  - Deploy the load balancer plug-in pod to the OpenShift cluster and allow cluster administrative permissions for the pod.
  - Deploy the router to support F5 BIG-IP load balancer.
- To enable IP failover in OpenShift Container Platform, deploy a deployment configuration within the cluster using the **oc adm ipfailover** command on all the nodes that require IP failover.
- To enable direct access using an external IP for any object type in OpenShift, configure the **networkConfig** section in each master node configuration file to point to the external IP address.
- OpenShift Container Platform dynamically manages **iptables** rules. Adding permanent rules may break OpenShift networking environment.