

Join the explorers, builders, and individuals who boldly offer new solutions to old problems. For open source, innovation is only possible because of the people behind it.

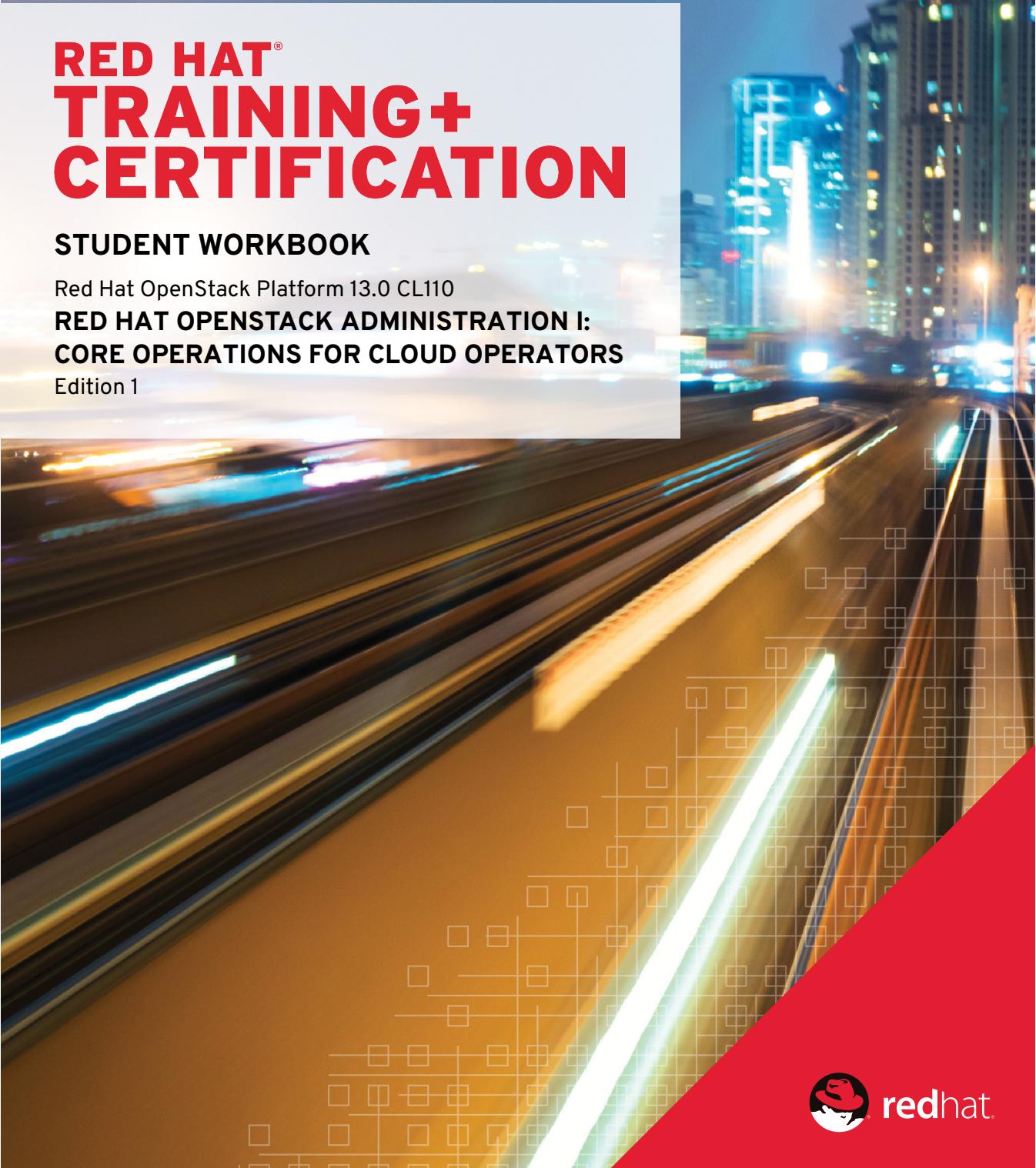
RED HAT® TRAINING + CERTIFICATION

STUDENT WORKBOOK

Red Hat OpenStack Platform 13.0 CL110

RED HAT OPENSTACK ADMINISTRATION I: CORE OPERATIONS FOR CLOUD OPERATORS

Edition 1



RED HAT OPENSTACK ADMINISTRATION I: CORE OPERATIONS FOR CLOUD OPERATORS



Red Hat OpenStack Platform 13.0 CL110
Red Hat OpenStack Administration I: Core Operations for
Cloud Operators
Edition 1 20180923
Publication date 20180923

Authors: Adolfo Vazquez, Chen Chang, Fiona Allen, Herve Quatremain,
Morgan Weetman, Snehangshu Karmakar
Editor: Philip Sweany, Seth Kenlon, David Sacco, Heather Charles

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

ix

Introduction

xi

Red Hat OpenStack Administration I	xi
Orientation to the Classroom Environment	xii
Controlling Your Systems	xix
Internationalization	xx

1. Introducing Launching an Instance

1

Launching an Instance Using the Dashboard	2
Workshop: Launching an Instance Using the Dashboard	8
Running the OpenStack Unified Command-line Interface	11
Guided Exercise: Running the OpenStack Unified CLI	19
Launching an Instance Using the CLI	24
Workshop: Launching an Instance Using the CLI	32
Describing the OpenStack Architecture	39
Quiz: Describing the OpenStack Architecture	45
Lab: Introducing Launching an Instance	49
Summary	55

2. Organizing People and Resources

57

Managing Projects	58
Guided Exercise: Managing Projects	62
Administering Users	64
Guided Exercise: Administering Users	70
Assigning User Roles and Privileges	75
Guided Exercise: Assigning User Roles and Privileges	79
Managing Quotas	84
Guided Exercise: Managing Quotas	89
Lab: Organizing People and Resources	94
Summary	101

3. Describing Cloud Computing

103

Describing Cloud Computing Concepts	104
Quiz: Describing Cloud Computing Concepts	109
Describing Virtual Machines and Containers	111
Quiz: Describing Virtual Machines and Containers	116
Illustrating Red Hat Cloud Product Use Cases	118
Quiz: Illustrating Red Hat Cloud Product Use Cases	122
Verifying OpenStack Services	124
Guided Exercise: Verifying OpenStack Services	129
Quiz: Describing Cloud Computing	133
Summary	137

4. Managing Linux Networks

139

Describing Networking Concepts	140
Quiz: Describing Networking Concepts	146
Managing Network Interfaces	150
Guided Exercise: Managing Network Interfaces	155
Implementing Linux Bridges	159
Guided Exercise: Implementing Linux Bridges	163
Implementing Open vSwitch Bridges	166
Guided Exercise: Implementing Open vSwitch Bridges	171
Lab: Managing Linux Networks	174
Summary	182

5. Preparing to Deploy an Instance

183

Uploading Images	184
Guided Exercise: Uploading Images	187

Developing Flavors	192
Guided Exercise: Developing Flavors	196
Managing Private Networks	199
Guided Exercise: Managing Private Networks	204
Lab: Preparing to Deploy an Instance	209
Summary	216
6. Deploying an Instance	217
Launching an Instance	218
Guided Exercise: Launching an Instance	221
Verifying the Functionality of an Instance	224
Guided Exercise: Verifying the Functionality of an Instance	235
Lab: Deploying an Instance	241
Summary	248
7. Managing Block Storage	249
Describing Features of the Cloud Storage Architecture	250
Quiz: Describing Features of the Cloud Storage Architecture	252
Managing Ephemeral Block Storage	256
Guided Exercise: Managing Ephemeral Block Storage	259
Administering Persistent Block Storage	263
Guided Exercise: Administering Persistent Block Storage	268
Developing Snapshots	274
Guided Exercise: Developing Snapshots	278
Managing Persistent Root Disks	288
Guided Exercise: Managing Persistent Root Disks	292
Lab: Managing Block Storage	299
Summary	311
8. Managing Object Storage	313
Describing Object Storage Architecture	314
Quiz: Describing Object Storage Architecture	319
Managing Objects	321
Guided Exercise: Managing Objects	329
Lab: Managing Object Storage	334
Summary	338
9. Preparing to Deploy an Instance with Public Access	339
Managing External Networks	340
Guided Exercise: Managing External Networks	343
Preparing OpenStack Routers	349
Guided Exercise: Preparing OpenStack Routers	354
Maintaining Floating IP Addresses	358
Guided Exercise: Maintaining Floating IP Addresses	361
Implementing Security	366
Guided Exercise: Implementing Security	377
Lab: Preparing to Deploy an Instance with Public Access	381
Summary	391
10. Deploying an Instance with Public Access	393
Launching an Instance with Public Access	394
Guided Exercise: Launching an Instance with Public Access	399
Verifying an Instance with Public Access	402
Guided Exercise: Verifying a Instance with Public Access	407
Managing Multi-tenant Networking	409
Quiz: Managing Multi-tenant Networking	414
Lab: Deploying an Instance with Public Access	416
Summary	421

11. Customizing Instances	423
Creating Customized Instances	424
Guided Exercise: Creating Customized Instances	428
Verifying Customized Instances	431
Guided Exercise: Verify Customized Instances	433
Lab: Customizing Instances	436
Summary	441
12. Deploying Scalable Stacks	443
Analyzing Cloud Metrics for Auto Scaling	444
Guided Exercise: Analyzing Cloud Metrics for Auto Scaling	456
Deploying a Stack	463
Guided Exercise: Deploying a Stack	478
Configuring Stack Auto Scaling	488
Guided Exercise: Configuring Stack Auto Scaling	497
Lab: Deploying Scalable Stacks	505
Summary	513
13. Deploying an OpenStack Overcloud	515
Creating Overcloud Deployment Plan Using Director Web UI	516
Guided Exercise: Creating an Overcloud Deployment Plan Using Director Web UI	525
Verifying the OpenStack Overcloud Deployment	529
Guided Exercise: Verifying the OpenStack Overcloud Deployment	532
Lab: Deploying an OpenStack Overcloud	548
Summary	568
14. Comprehensive Review	569
Comprehensive Review	570
Lab: Deploying an FTP Server	573
Lab: Deploying a Web Server	598
Lab: Deploying OpenStack and Launching a Stack	616

DOCUMENT CONVENTIONS



REFERENCES

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



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.

INTRODUCTION

RED HAT OPENSTACK ADMINISTRATION I

Red Hat OpenStack Administration I (CL110) is designed for system administrators who are intending to implement a cloud computing environment using OpenStack. Students will learn how to install, configure, use, and maintain Red Hat OpenStack Platform.

The focus of this course will be managing OpenStack using the Horizon dashboard and the command-line interface, managing instances, and installing a proof of concept. Exam competencies covered in the course include: install and configure Red Hat OpenStack Platform (using PackStack), manage users, projects, flavors, roles, images, networking, and block storage, set quotas, and configure instances at instantiation.

COURSE OBJECTIVES

- Launch and customize instances in a private OpenStack cloud.
- Manage core OpenStack services using the Horizon web interface and the command-line interface.
- Deploy scalable stacks.
- Install a simple proof-of-concept OpenStack deployment.

AUDIENCE

- Cloud administrators, cloud operators, and system administrators interested in, or responsible for, maintaining a private cloud.

PREREQUISITES

- Red Hat Certified System Administrator (RHCSA in RHEL) certification or equivalent experience.

ORIENTATION TO THE CLASSROOM ENVIRONMENT

Figure 0.1: Classroom environment

In this classroom environment, the primary student system for all hands-on activities is **workstation**. The **workstation** virtual machine (VM) is the only one with a graphical desktop, required for browser access to remote dashboard and GUI tools. Students are instructed to always first log in directly to the **workstation** VM. From **workstation**, use SSH to access all other VMs for command line use. Use a web browser on **workstation** to access the Red Hat OpenStack Platform Dashboard web interface and other graphical UI tools.

As seen in Figure 0.1, all VMs share an external network, 172.25.250.0/24, with a gateway of 172.25.250.254 (**workstation**). External network DNS services are also provided by **workstation**. The overcloud virtual machines share an internal network containing multiple VLANs, using various 172.24.X.0/24 addresses.

Additional student VMs used for hands-on exercises include **director** and **power** in the **lab.example.com** DNS domain, and **controller0**, **compute0**, **compute1**, and **ceph0** in the **overcloud.example.com** DNS domain. The overcloud VMs share the provisioning network, **172.25.249.0/24** with the **director** and **power** nodes. The undercloud uses the isolated and dedicated provisioning network to deploy the overcloud nodes.

The environment uses the **classroom** server as a NAT router to the outside network, and as a file server using the URLs **content.example.com** and **materials.example.com**, serving course content for certain exercises. The **workstation** VM is also a router to the classroom network, and *must remain running for proper operation of all other VMs*.

Classroom Machines

MACHINE NAME	IP ADDRESSES	ROLE
workstation.lab.example.com , workstationN.example.com	172.25.250.254 172.25.252.N	Graphical student workstation
director.lab.example.com	172.25.250.200 172.25.249.200	Standalone undercloud node as director
power.lab.example.com	172.25.250.100 172.25.249.100 172.25.249.101+	Handles overcloud node IPMI power management
controller0.overcloud.example.com	172.25.250.1 172.25.249.P 172.24.X.1	An unclustered overcloud controller node
compute0.overcloud.example.com	172.25.250.2 172.25.249.R 172.24.X.2	An overcloud compute node

MACHINE NAME	IP ADDRESSES	ROLE
<code>compute1.overcloud.example.com</code>	172.25.250.12 172.25.249.S 172.24.X.12	Another overcloud compute node
<code>ceph0.overcloud.example.com</code>	172.25.250.3 172.25.249.T 172.24.X.3	The overcloud block and object storage node
<code>classroom.example.com</code>	172.25.254.254 172.25.253.254 172.25.252.254	Classroom materials server

The **workstation** VM uses a **student** user with the password **student**. The **director** VM uses a default **stack** user with the password **redhat**. The **root** password on most VMs is **redhat**. These overcloud nodes were preconfigured with a **heat-admin** account, used by the deployment service to configure these nodes. Access to overcloud nodes is by key-based passwordless SSH access from **workstation** or **director**.

System and Application Credentials

SYSTEM CREDENTIALS	USERNAME	PASSWORD
Unprivileged shell login (as directed)	student	student
Privileged shell login (as directed)	root	redhat
Undercloud node unprivileged access	stack	redhat
Undercloud node privileged access	root	redhat
Overcloud node unprivileged access	heat-admin	passwordless SSH
Overcloud node privileged access	root	use 'sudo -i'
APPLICATION	USERNAME	PASSWORD
Red Hat OpenStack Platform dashboard admin	admin	redhat
Red Hat OpenStack Platform dashboard user	as directed	redhat
Red Hat OpenStack Platform director	admin	redhat

CLASSROOM NETWORK TOPOLOGY

The following Figure 0.2 shows the infrastructure design.

Figure 0.2: CL110 classroom network topology

MANAGING OPENSTACK OVERCLOUDS IN THE CLASSROOM

OpenStack overclouds require continuous communication and coordination between the undercloud director node, the power node, and overcloud nodes. OpenStack systems must be started and stopped in the correct order using recommended commands and actions, so that

nodes are not left in an unresponsive or corrupt state. *This discussion focuses on a classroom-based behavior only*, because production OpenStack installations are highly available, redundant, resilient, and scalable infrastructures that *never need to be shut down*.

This course includes a custom **rht-overcloud** systemd service installed on the **director** node, which automatically starts the overcloud nodes in the correct order when the student classroom environment is first started each day. The **rht-overcloud** is not currently configured to perform a similar service for stopping the overcloud at the end of the day or work session. Instead, students should practice controlling an overcloud using the Red Hat supported procedures recommended for production overclouds, as described below.



NOTE

The **rht-overcloud** service is not intended to be invoked manually.

Properly Shutting Down a Red Hat OpenStack Deployment

Red Hat OpenStack Platform is a cloud infrastructure architect for perpetual operation. Except for extraordinary maintenance scenarios, underclouds and overclouds are never expected to be shutdown altogether. Controller nodes are installed using high availability tools, compute nodes are configured in quantities designed to be expendable and redundant, and storage nodes use software that is also designed for redundant, replicated scaling. Single controller, compute and storage nodes may be temporarily disabled for maintenance, and not affect the safe operations of the remainder of the OpenStack cloud.

Although the procedures described here are infrequently needed in production, the transient nature of a classroom environment requires daily startup and shutdown.

- Before the overcloud can be shut down, all project workloads must first be stopped. Stop (not terminate) the instances running on the compute nodes in the overcloud. To list all the instances running on the compute nodes, use the following command:

```
[student@workstation ~]$ ssh director
[stack@director ~]$ source ~/overcloudrc
[stack@director ~]$ openstack server list --all-projects
+-----+-----+-----+-----+
| ID           | Name      | Status   | Networks    | Image
|             |
+-----+-----+-----+-----+
| ac0ddb44-996f-4710-84c0-a78d3261c362 | finance31 | ACTIVE   | network1 ... | rhel7
|             |
| 7051e207-99b9-4652-808c-7f50418b4a79 | research1 | ACTIVE   | network2 ... | rhel7
|             |
| 895e18e1-40b4-7201-4b4b-6c479823ab4a | consult28 | ACTIVE   | network3 ... | rhel7
|             |
+-----+-----+-----+-----+
```

- Use the instance UUID from the output of the above command to stop the instances:

```
[stack@director ~]$ openstack server stop ac0ddb44-996f-4710-84c0-a78d3261c362
[stack@director ~]$ openstack server stop 7051e207-99b9-4652-808c-7f50418b4a79
[stack@director ~]$ openstack server stop 895e18e1-40b4-7201-4b4b-6c479823ab4a
```

- Once the instances are stopped, proceed with shutting down the overcloud compute nodes.

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ openstack server list
+-----+-----+-----+
| ID           | Name      | Status | Networks
+-----+-----+-----+
| 8d97b6a0-8989-4bbd-bdaf-b150327e | compute0   | ACTIVE | ctlplane=172.25.249.56
| 5c1cb371-9cf0-4ef4-a6a5-7c907c7a | compute1   | ACTIVE | ctlplane=172.25.249.52
| c5572e04-9be3-4d58-bfed-9153a80c | controller0 | ACTIVE | ctlplane=172.25.249.55
| fe4d8569-1713-490e-96c0-200de700 | ceph0      | ACTIVE | ctlplane=172.25.249.59
+-----+-----+-----+
```

- Log in to each compute node and trigger poweroff:

```
[stack@director ~]$ ssh compute0
[heat-admin@compute0 ~]$ sudo poweroff
Connection to compute0 closed by remote host.
Connection to compute0 closed.
[stack@director ~]$ ssh compute1
[heat-admin@compute1 ~]$ sudo poweroff
Connection to compute1 closed by remote host.
Connection to compute1 closed.
```



NOTE

The computes will still show as **up** from undercloud until they are polled by nova to sync the displayed power state. This can take many minutes or not at all if the sync routine is disabled. Do not wait, as the nodes will be actually powered off.



CAUTION

In the classroom environment, **controller0** is the Ceph admin node. To shut down Ceph properly, set the Ceph flags from **controller0**, before logging in to **ceph0** to trigger poweroff, then return to **controller0** to stop the Pacemaker cluster. Stopping the cluster first will improperly disable admin control of the Ceph server.

- Set the **noout**, **norecover**, **norebalance**, **nobackfill**, **nodown** and **pause** flags.

```
[stack@director ~]$ ssh controller0
[heat-admin@controller0 ~]$ ceph osd set noout
noout is set
[heat-admin@controller0 ~]$ ceph osd set norecover
norecover is set
[heat-admin@controller0 ~]$ ceph osd set norebalance
```

```
norebalance is set  
[heat-admin@controller0 ~]$ ceph osd set nobackfill  
nobackfill is set  
[heat-admin@controller0 ~]$ ceph osd set nodown  
nodown is set  
[heat-admin@controller0 ~]$ ceph osd set pause  
pauserd,pausewr is set
```

- Log in to the Ceph0 node and trigger poweroff:

```
[stack@director ~]$ ssh ceph0  
[heat-admin@ceph0 ~]$ sudo poweroff  
Connection to ceph0 closed by remote host.  
Connection to ceph0 closed.
```

- Remaining OpenStack services need to be stopped, to ensure system and data integrity. On the controller node, issue the following command and wait for the cluster to stop:

```
[heat-admin@controller0 ~]$ sudo pcs cluster stop --all  
controller0: Stopping Cluster (pacemaker)...  
controller0: Stopping Cluster (corosync)...  
[heat-admin@controller0 ~]$ sudo poweroff  
Connection to controller0 closed by remote host.  
Connection to controller0 closed.
```

- The overcloud is now shut down. To shut down the full Red Hat OpenStack deployment, also power off the undercloud system. If the undercloud is shut down, restarting will require using classroom-environment-specific instructions to start the **director** system.

```
[stack@director ~]$ sudo shutdown -h now  
Connection to director closed by remote host.  
Connection to director closed.  
[student@workstation ~]$
```

When finished with class for the day, or when you need a longer break from practice, the recommendation is to cleanly shut down the classroom environment.

From **foundationX**, use the **rht-vmctl** command to stop the remaining virtual machines.

```
[kiosk@foundationX ~]$ rht-vmctl stop all
```

Properly Starting a Red Hat OpenStack Deployment

Most services in Red Hat OpenStack Platform are designed to start and wait for their dependencies, even if started out of order. However, starting the nodes in the recommended order will minimize possible race conditions and startup delays.

- Before starting the overcloud, the undercloud must be running. Power on the **director** system using the classroom-specific instructions. When the undercloud is available, log in to the **director** system. Verify that the **nova-compute** service is **up**.

```
[student@workstation ~]$ ssh director  
[stack@director ~]$ openstack compute service list -c Binary -c Status -c State  
+-----+-----+-----+
```

Binary	Status	State
nova-conductor	enabled	up
nova-scheduler	enabled	up
nova-compute	enabled	up



NOTE

This procedure lists manual steps for starting the overcloud. In the classroom environment, powering on the **director** node invokes the custom **rht-overcloud** systemd service to automatically power on overcloud nodes. Therefore, some steps may be skipped if the node affected is already started.

- Check the power state of all nodes. Start the **ceph0** node first if NOT already powered on.

```
[stack@director ~]$ openstack baremetal node list -c Name -c "Power State"
+-----+-----+
| Name      | Power State |
+-----+-----+
| controller0 | power on   |
| compute0    | power on   |
| compute1    | power on   |
| ceph0       | power on   |
+-----+-----+
[stack@director ~]$ openstack baremetal node power on ceph0
```

- Start the **controller0** node if NOT already powered on, then log in and start the cluster.

```
[stack@director ~]$ openstack baremetal node power on controller0
[stack@director ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo pcs cluster start --all
controller0: Starting Cluster...
```

- On **controller0**, unset **noout**, **norecover**, **norebalance**, **nobackfill**, **nodown** and **pause** flags to allow the Ceph storage server to handle normal recovery tasks again.

```
[heat-admin@controller0 ~]$ ceph osd unset noout
noout is unset
[heat-admin@controller0 ~]$ ceph osd unset norecover
norecover is unset
[heat-admin@controller0 ~]$ ceph osd unset norebalance
norebalance is unset
[heat-admin@controller0 ~]$ ceph osd unset nobackfill
nobackfill is unset
[heat-admin@controller0 ~]$ ceph osd unset nodown
nodown is unset
[heat-admin@controller0 ~]$ ceph osd unset pause
pauserd, pausewr is unset
[heat-admin@controller0 ~]$ exit
```

- Once the cluster is back online, power on the overcloud compute nodes if NOT already started.

```
[stack@director ~]$ openstack baremetal node power on compute0  
[stack@director ~]$ openstack baremetal node power on compute1
```

- Proceed to start any desired project server instances that were previously shut down.

The overcloud is now fully started.

TO RESET THE CLASSROOM ENVIRONMENT

Critical concept

When the CL110 coursebook includes instructions to reset virtual machines, the intention is to reset only the overcloud nodes to an initial state. Unless something else is wrong with any physical system or online environment that is deemed irreparable, there is no reason to reset all virtual machines or to reprovision a new lab environment.

What it means to reset the overcloud

Whether you are working in a physical or online environment, certain systems never need to be reset because they remain materially unaffected by exercises and labs. This table lists the systems never to be reset and those intended to be reset as a group during this course:

Which systems normally should or should not be reset

NEVER NEEDING TO BE RESET	ONLY RESET TOGETHER AS A GROUP
• classroom	• controller0
• workstation	• compute0
• power	• compute1
	• ceph0
	• director

Technically, the **director** system is the undercloud. However, in the context of resetting the overcloud, **director** must be included because director's services and databases are full of control, management, and monitoring information about the overcloud it is managing. Therefore, to reset the overcloud without resetting **director** is to load a fresh overcloud with **director** retaining stale information about the previous overcloud that was just discarded.

In a physical classroom, use the **rht-vmctl** command to reset only the relevant nodes. There are group names built into the **rht-vmctl** command for this course: **undercloud** and **overcloud**. Use these group names easily manage the system groupings with single commands. For example, to reset both the undercloud and overcloud systems, run these commands:

```
[kiosk@foundation ~]$ rht-vmctl reset undercloud  
[kiosk@foundation ~]$ rht-vmctl reset overcloud
```

OPENSTACK PACKAGES AND DOCUMENTATION

Repositories suitable for RPM package installation are available locally in your environment at http://content.example.com/rhosp13.0/x86_64/dvd/.

Software documentation is available at <http://materials.example.com/docs/>, which contains subdirectories for files in PDF and single-page HTML format.

CONTROLLING YOUR SYSTEMS

In an instructor-led training classroom, students are assigned a physical computer (**foundationX.ilt.example.com**), which is used to access their virtual machines running on that host. Students are automatically logged in to the physical machine as the **kiosk** user, with the password **redhat**. The classroom systems with which you work are virtual machines running on that host.

On **foundationX**, a special command called **rht-vmctl** is used to work with the virtual machines. The commands in the following table should be run as the **kiosk** user on **foundationX**, and can be used with any of the virtual machines.

rht-vmctl Commands

ACTION	COMMAND
Start server machine	rht-vmctl start server
View "physical console" to log in and work with the server machine	rht-vmctl view server
Reset server machine to its previous state and restart the virtual machine	rht-vmctl reset server

At the start of a lab exercise, if the instruction "reset your servera" appears, you should run the **rht-vmctl reset servera** command on the **foundationX** system as the **kiosk** user. Likewise, if the instruction "reset your workstation" appears, you should run the **rht-vmctl reset workstation** command on **foundationX** as the **kiosk** user.

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. Click the Language box and select the preferred language from the list that appears. This also updates 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, and 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 determine 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 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.

When 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 also find 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, the **root** user can change the system-wide locale settings with the **localectl** command. If **localectl** is run with no arguments, it displays 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 language packs, run **yum langavailable**. To view the list of language packs currently installed on the system, run **yum langlist**. To add an additional language pack 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

LANGUAGE	\$LANG VALUE
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

INTRODUCING LAUNCHING AN INSTANCE

GOAL

Launch an instance, and describe the OpenStack architecture and use cases.

OBJECTIVES

- Launch an instance in the dashboard given a preconfigured OpenStack installation.
- Describe the identity environment file and run the unified command-line interface.
- Launch an instance using the command-line interface.
- Describe the OpenStack architecture and use cases.

SECTIONS

- Launching an Instance Using the Dashboard (and Guided Exercise)
- Running the OpenStack Unified CLI (and Guided Exercise)
- Launching an Instance Using the CLI (and Guided Exercise)
- Describing the OpenStack Architecture (and Quiz)

LAB

- Launching an Instance

LAUNCHING AN INSTANCE USING THE DASHBOARD

OBJECTIVES

After completing this section, students should be able to launch an instance using the dashboard given a preconfigured OpenStack installation.

LOGGING IN TO THE DASHBOARD WEB INTERFACE

The Red Hat OpenStack Platform dashboard is a browser-based graphical user interface for managing OpenStack services. This dashboard can be customized.

Accessing the dashboard requires a URL, user account, and password. It is accessible over HTTP or HTTPS, for example: <http://dashboard.overcloud.example.com>.

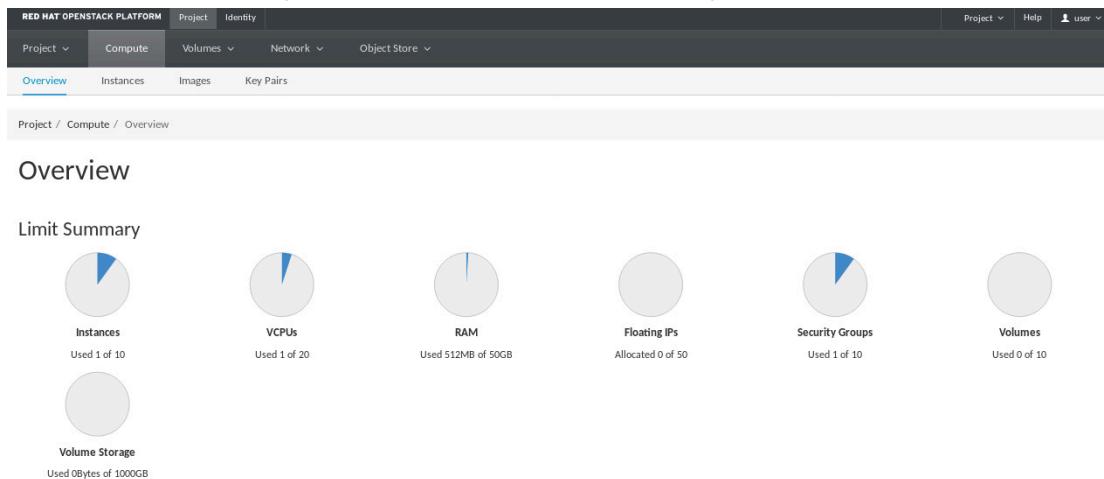


Figure 1.1: The OpenStack dashboard

WORKING WITH SELF-SIGNED CERTIFICATES

A certificate authority (CA) can be used to generate the certificates used by OpenStack, including the web server certificate for the dashboard. In the classroom, the default certificate mechanisms are used, which generate a self-signed certificate. Firefox does not accept these self-signed certificates by default because of the security implications. To manually accept the certificate, perform the following steps:

1. Browse to the dashboard URL.
2. Expand the I Understand the Risks menu.
3. Click Add Exception.
4. Click Confirm Security Exception to permanently store this exception.

WORKING WITH USERS IN A PROJECT

OpenStack users are self-service users. They can create and manage instances, volumes, networks, and images. Assigning users to a *project* limits their permissions and the amount of resources they can access. Additionally, OpenStack users can be assigned specific roles, which gives them a limited set of rights and privileges.

The OpenStack self-service mechanism allows administrators to delegate tasks to users. It allows them to manage the projects they are assigned. Self-service users can create resources,

or instances, and use the available infrastructure without the need to contact administrators. This benefits users and administrators, because it avoids the latencies involved when these two types of users have to coordinate. This reduces the time, and processes needed to be defined and completed, for a project to be deployed. It also helps developers to deploy their applications more quickly and easily when testing different architectures for their applications.

The Identity tab in the dashboard provides an interface for viewing and managing projects and users. When creating an OpenStack cloud, *project* acts as a container of resources owned by different users. A set of resource quotas is configured when a new project is created. The quotas include the number of instances, number of VCPUs, amount of RAM, and floating IPs that can be assigned to instances within the project. Users can be assigned to multiple projects, but one of the projects is designated as the *primary project*. The primary project is simply the first project to which a user is associated.

You can use the dashboard to create, modify, and delete projects. You can also use the dashboard to view project usage, and add or remove users as project members, modify quotas, and set a project active. Users with the **admin** role, and associated with the **admin** project, can use the dashboard to view, create, edit, and delete users, and change their passwords. The Users tab is only visible if a user is logged in with administrative privileges. Users can have access to multiple projects. The actions users can perform in a project are defined in the *user roles*. OpenStack ships with two predefined roles, **_member_** and **admin**. Adding a user with the **admin** role promotes the user to an administrative user. You can use the dashboard to create or edit *user roles*. A role defines a set of rights and privileges. When a role is assigned to a user, that user inherits the rights and privileges of that role. When a user requests a service, that service interprets the user role assigned to the user and provides access based on the role.



NOTE

In earlier versions of Red Hat OpenStack Platform, the term *projects* was used with command-line utilities, and the term *projects* was introduced with the dashboard. Current versions of OpenStack use the term *projects* in both the dashboard and the unified command-line interface.

The screenshot shows the Red Hat OpenStack Platform dashboard with the 'Identity' tab selected. The main navigation bar includes 'Project' and 'Identity'. Below the navigation, there are tabs for 'Projects' and 'Users', with 'Projects' currently active. The breadcrumb navigation shows 'Identity / Projects'. The main content area is titled 'Projects' and displays a table with one item. The table columns are 'Name', 'Description', 'Project ID', 'Domain Name', 'Enabled', and 'Actions'. The single row shows 'demo' as the name, 'Demonstration' as the description, 'b3e2601ab38544da8835c915bb1f6613' as the Project ID, 'Default' as the Domain Name, and 'Yes' under 'Enabled'. There is a 'Filter' button at the bottom right of the table. The status bar at the bottom indicates 'Displaying 1 item'.

Figure 1.2: Projects in the dashboard

COMPONENTS REQUIRED TO LAUNCH AN INSTANCE

The OpenStack term *instance* refers to a virtual machine instance. The dashboard can launch and manage an instance. Prior to launching an instance, you need to create and upload an image, which constitutes the operating system and software for the instance.

Launching an Instance

In a traditional virtualization environment, several steps must be performed before a new virtual system can be used.

- Virtual hardware needs to be defined.
- Storage for the VM needs to be defined.
- An operating system need to be installed.

These steps take time and physical resources.

In a Red Hat OpenStack Platform environment, the virtual machines are stored as images, which provide templates for virtual machine file systems. When a new virtual machine is needed, a new instance is launched. Many instances can be launched from the same image.

- An instance is booted from a copy of an original image containing a preinstalled operating system.
- The compute node copies the original image and boots the instance from the copy (called the *base image*).
- Live changes made to the instance are stored in an overlay file so as to leave the base image unaffected during use.
- When the instance is deleted, memory and vCPU resources are released. The original state is reclaimed by deleting the overlay file which contained all the runtime changes.
- When the instance is deleted, any persistent volumes and the base image are retained and remain unchanged, and are available for reuse.

This mechanism allows for a flexible and rapid deployment of virtual instances.

Images

An image is a file that contains a virtual disk with a bootable operating system installed. Images can either be created or customized using the dashboard. Several prebuilt images provided by various software vendors can also be imported. In a multiple project cloud environment, users can also share their personal images with other projects. These images can be of various formats (RAW, QCOW2, ISO, VMDK, VHD, ARI, AKI, AMI, PLOOP, Docker, OVA and VDI) which can be imported in to OpenStack.

Images for the current project can be found in the dashboard under **Images**. To create a new image, click **Create Image**. Images for the current project can be modified in the dashboard under **Images**. To modify images, expand the **Launch** menu and click **Edit Image**.

	Name	Type	Status	Visibility	Protected	Disk Format	Size	Actions
<input type="checkbox"/>	rhe7	Image	Active	Public	No	RAW	623.15 MB	<button>Launch</button> <button>Edit</button>

Figure 1.3: Images in the dashboard

Flavors

Virtual hardware templates are known as *flavors* in OpenStack. Flavors specify the minimum amount of RAM, minimum disk space, and the number of vCPUs. The dashboard provides the ability to modify and delete an existing flavor, or create a new one.

Flavors for the current project can be found in the dashboard when an unprivileged user launches an instance. Only administrators have permission to edit or create new flavors (the rights may also

be delegated to other users). An administrator account has access to the Admin tab which provides Flavors.

Networks

Networks define the network resources to be added to an instance. Based on the instance's network and the routing table, the instance can be isolated or have external, public access. Administrators can configure robust network topologies by creating and configuring networks and subnets, and instruct other OpenStack services to attach virtual devices to ports on these networks and instances. OpenStack networking supports multitenancy; the creation of multiple private networks for each project, and allowing projects to choose their own IP addressing scheme, even if those IP addresses overlap with those used by other projects.

The network topology can be viewed using the dashboard under the Project menu, or click Project → Network → Network Topology.

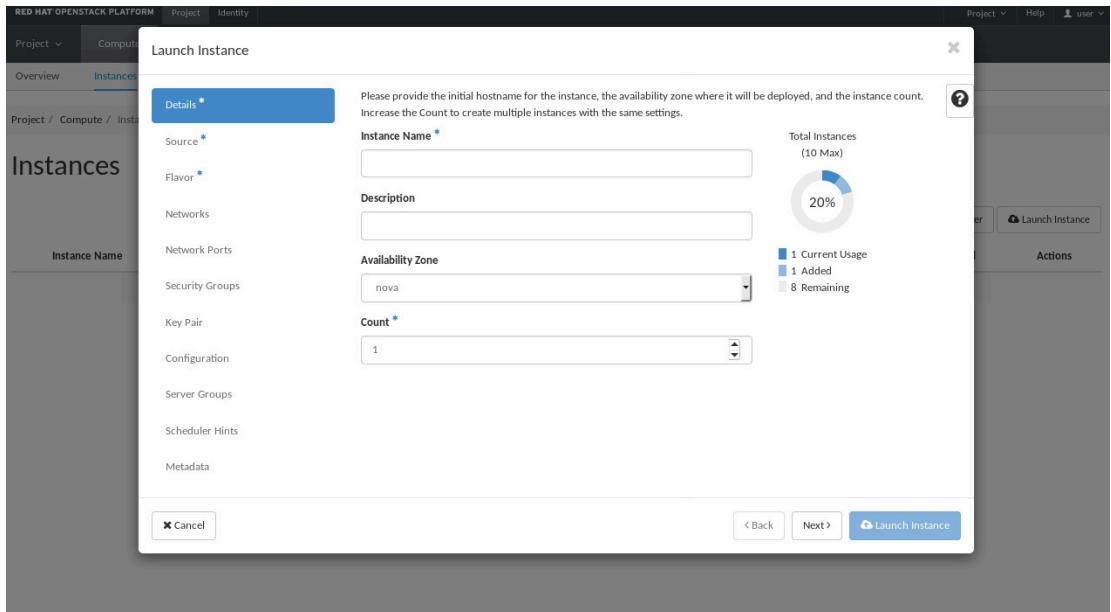


Figure 1.4: Launching an instance in the dashboard

CONNECTING TO AN INSTANCE

Connecting to the Instance Console

After a successful instance launch, the dashboard can connect to the instance. The console provides a way to directly access the instance. In the dashboard, click Compute and then click Instances. Choose the line with the instance you want to connect to, click More, and then select Console. Log in to the instance console using a valid user name and password for the operating system running on the instance.

VERIFYING THE INSTANCE

To verify the setup, log in to the instance and try to reach other instances using the network.

Verifying Outside Connections

The Red Hat OpenStack Platform Networking DHCP agent manages the network namespaces that are spawned for each project subnet to act as a DHCP server. Each namespace runs a **dnsmasq** process that is capable of allocating IP addresses to virtual machines running on the network. If the agent is enabled and running when a subnet is created, then that subnet has DHCP enabled by default. The DHCP server IP address in a particular subnet can be found using the dashboard.

To determine the DHCP server IP address used in a network, click the Network tab and then click the Networks link. On the next window, click the name of the network. On the Network Overview page, click the Ports tab. The DHCP server IP address is listed under Fixed IPs.

By default the DHCP server can be pinged from the instance network. Use the **ping -c3 X.X.X.X** command to try to reach the DHCP server. The output will be similar to the following:

```
[user@demo ~]$ ping -c3 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.642 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.457 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.596 ms
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.457/0.565/0.642/0.078 ms
```

ACCESSING THE CONSOLE LOG

If there is an error, or the behavior of an instance should be checked, a self-service user can use the dashboard to access the console log of an instance.

Viewing the Console Log

To view the console log, right-click the instance link and choose Open Link in New Tab. In the new tab, choose the Log tab.

In the newly opened window, view the console log which includes messages generated by different services, including the **cloud-init** service. Possible errors and the current configuration can be identified, for instance the network routing table. The following is an example of a console log from a running instance:

```
...
[[32m OK [0m] Reached target Login Prompts.
[[32m OK [0m] Started Command Scheduler.
    Starting Command Scheduler...
[ 41.324203] cloud-init[493]: Cloud-init v. 0.7.6 running 'init-local' . Up
41.08 seconds.
[[32m OK [0m] Started Network Manager Wait Online.
    Starting LSB: Bring up/down networking...

Red Hat Enterprise Linux Server 7.5 (Maipo)
Kernel 3.10.0-862.el7.x86_64 on an x86_64

small image
demo-server1 login:
cloud-init[805]: Cloud-init v. 0.7.6 running 'init' Up 50.94 seconds.
```

The console log output includes information about the operating system, the running kernel version, and different messages from the boot process. It can also be used for debugging various problems, such as networking issues. The log includes detailed information about the status of the network services, interfaces, IP addresses, and so on. A system administrator can use this information to help determine the cause of possible problems, without the need to log in to the instance itself.



REFERENCES

Further information is available in the Virtual Machine Instances section of the *Instances and Images Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/instances_and_images_guide/

LAUNCHING AN INSTANCE USING THE DASHBOARD

In this exercise, you will launch an instance using the OpenStack dashboard.

OUTCOMES

You should be able to:

- Log in to the dashboard.
- Launch an instance using the dashboard.
- Verify that the instance is running properly.

BEFORE YOU BEGIN

Confirm the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **launch-instance** script with the **setup** argument. It creates the required components for this exercise, such as users, passwords, images, and networks.

```
[student@workstation ~]$ lab launch-instance setup
```

- ▶ 1. On **workstation**, open Firefox and browse to `http://dashboard.overcloud.example.com`.
- ▶ 2. Log in to the dashboard using **developer1** as the user name and **redhat** as the password.
- ▶ 3. Click the Project menu to verify that **finance** is the current project.
- ▶ 4. Launch an instance named **finance-server1** using the **default** flavor, the **rhe17** image, and the **finance-network1** network.
 - 4.1. Click the Instances subtab under the Compute tab.
 - 4.2. Click Launch Instance.
 - 4.3. In the Details tab, enter **finance-server1** as the Instance Name.
 - 4.4. On the Source tab, select Select Boot Source → Image.
 - 4.5. On the Source tab, click the up arrow button on the same row as the image to allocate the **rhe17** image. Click No under Create New Volume.
 - 4.6. On the Flavor tab, click the up arrow button on the same row as the flavor to allocate the **default** flavor.
 - 4.7. On the Networks tab, if the **finance-network1** network is not already allocated, click the up arrow button on the same row as the network to allocate the **finance-network1** network.

4.8. Click Launch Instance to launch **finance-server1**.

- ▶ 5. Wait a few seconds and then verify the status of the instance in the Power State column. The instance should be in a **Running** state.
- ▶ 6. To view the console, right-click the finance-server1 instance link and choose Open Link in New Tab. In the new tab, choose the Console tab, then click the Click here to show only console link. If a certificate error appears, accept the self-signed certificate. Watch the virtual machine boot (it may have already booted).
- ▶ 7. To verify the setup, log in to the **finance-server1** instance as the **root** user with **redhat** as the password.

```
small image
finance-server1 login: root
Password: redhat
[root@finance-server1 ~]#
```

- ▶ 8. List the routing table to find the **gateway** server IP address.

```
[root@finance-server1 ~]# ip route show
default via 192.168.1.1 dev eth0 proto dhcp metric 100
169.254.169.254 via 192.168.1.2 dev eth0 proto dhcp metric 100
192.168.1.0/24 dev eth0 proto kernel scope link src 192.168.1.X metric 100
```

- ▶ 9. Verify that the gateway (**192.168.1.1** in the previous example output) server is reachable from **finance-server1**.

```
[root@finance-server1 ~]# ping -c3 192.168.1.1
PING 192.168.1.1 (192.168.1.1) 56(84) bytes of data.
From 192.168.1.3: icmp_seq=1 Destination Host Unreachable
From 192.168.1.3: icmp_seq=2 Destination Host Unreachable
From 192.168.1.3: icmp_seq=3 Destination Host Unreachable
--- 192.168.1.1 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 1000ms
```

At this point, no external router is attached to this network, so it does not have an active gateway and you cannot ping any address outside of the 192.168.1.0/24 network.

- ▶ 10. Verify that the DHCP server is reachable from **finance-server1**. The DHCP server IP address in this exercise is **192.168.1.2**.

In the instance console, verify the connectivity using the **ping** command with the IP address of the DHCP server, **192.168.1.2**.

```
[student@finance-server1 ~]$ ping -c3 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.642 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.457 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.596 ms
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.457/0.565/0.642/0.078 ms
```

- 11. Close the instance tab in the browser. Log out of the dashboard by clicking the developer1 menu in the upper-right corner, then choose Sign out.

Cleanup

On **workstation**, run the **lab launch-instance cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab launch-instance cleanup
```

This concludes the guided exercise.

RUNNING THE OPENSTACK UNIFIED COMMAND-LINE INTERFACE

OBJECTIVES

After completing this section, students should be able to:

- Describe the identity environment file and run the OpenStack client unified command-line interface (CLI).
- View Red Hat OpenStack Platform resources using the OpenStack client CLI.

RED HAT OPENSTACK PLATFORM TOOLSET

The dashboard provides a user-friendly environment for cloud users and administrators, but some tasks may require more flexibility, such as scripting capabilities, to support operations. Red Hat OpenStack Platform includes the OpenStack client CLI that is designed to implement all the common, necessary functionality offered by the OpenStack services APIs. As OpenStack core and extended service are enhanced, new functionality is added to the unified CLI.

Early Red Hat OpenStack Platform versions implemented this CLI toolset as a group of client tools, one for each OpenStack service. For example, the **keystone** command managed the identity service for OpenStack, and the **nova** command managed the compute service. These python-based client tools are now re-architected into a single, unified OpenStack client CLI. This simplifies operations within the Red Hat OpenStack Platform environment and uses the single **openstack** command with resource object specific commands.

IDENTITY SERVICE CREDENTIALS

In the same way that the dashboard requires an authenticated user name and password, the OpenStack CLI requires user authentication before using Red Hat OpenStack Platform environment services. For successful authentication, users must specify, as a minimum, the parameters described in the following table. The parameters can be provided as either environment variables or **openstack** command arguments.

Identity service authentication parameters

ENVIRONMENT VARIABLE	OPENSTACK COMMAND ARGUMENT	DESCRIPTION
OS_USERNAME	--os-username	User's user name
OS_PASSWORD	--os-password	User's password
OS_PROJECT_NAME	--os-project-name	Project to which the user belongs
OS_PROJECT_DOMAIN_NAME	--os-project-domain-name	Domain name to which the project belongs
OS_USER_DOMAIN_NAME	--os-user-domain-name	Domain name to which the user belongs
OS_IDENTITY_API_VERSION	--os-identity-api-version	Identity API version

ENVIRONMENT VARIABLE	OPENSTACK COMMAND ARGUMENT	DESCRIPTION
OS_AUTH_URL	--os-auth-url	Public URL to the Identity API endpoint



NOTE

The end point included in the **OS_AUTH_URL** environment variable corresponds to the public end point for the Identity service. By default, this is bound to TCP port 5000.

If authentication parameters are specified using environment variables, Red Hat recommends that a **<username>-<project>-rc** file is created for each user's authentication parameters per project. For example, a user named **user** that belongs to the **demo** project, with **redhat** as a password, and the identity service end point located on the **172.25.250.50** server, could use the following environment file named **user-demo-rc**:

```
export OS_USERNAME=user
export OS_PASSWORD=redhat
export OS_PROJECT_NAME=demo
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_URL=http://172.25.250.50:5000/v3
```

Source the **user-demo-rc** file to set the required environment variables.

```
[user@demo ~]$ source user-demo-rc
```

After sourcing the identity environment file, the user can run the **openstack** command without requiring to include authentication parameters as command options. Without using the identity environment file, a user would specify authentication parameters as options on the **openstack** command. For example, the following command would list the images available in the Image service using the same authentication as described previously:

```
[user@demo ~]$ openstack \
--os-username user --os-password redhat --os-project-name demo \
--os-project-domain-name Default --os-user-domain-name Default \
--os-identity-api-version 3 --os-auth-url http://172.25.250.50:5000/v3 \
image list
```



NOTE

By default, Red Hat OpenStack Platform director creates a file named **overcloudrc** file in the **/home/stack** directory on the undercloud system. It contains the administrator's authentication parameters for the deployed overcloud Red Hat OpenStack Platform environment.

Accessing Red Hat OpenStack Platform with a Identity Service Environment File

Use the following steps to create an identity environment file to gain access rights within the overcloud Red Hat OpenStack Platform environment:

1. Create a file named **~/user-demo-rc**.
2. Modify the file to contain the authentication parameters for the **user** user, including the following variables:
 - **OS_USERNAME**: specifies the **user** user.
 - **OS_PASSWORD**: specifies the user's password.
 - **OS_PROJECT_NAME**: specifies the project name for **user**.
 - **OS_PROJECT_DOMAIN_NAME**: specifies the project's domain name for **user**.
 - **OS_USER_DOMAIN_NAME**: specifies the user's domain name for **user**.
 - **OS_IDENTITY_API_VERSION**: specifies the identity service API version to **3**.
 - **OS_AUTH_URL**: specifies the identity service public endpoint URI.
3. Source the **user-demo-rc** file to gain access rights within the overcloud Red Hat OpenStack Platform environment.

IDENTITY SERVICE TOKENS

To enhance security, OpenStack uses a unique access code, or token, to authenticate requests to OpenStack services APIs. User authentication and authorization is controlled by the *Identity* service (keystone). This service securely checks the user's identity and generates a unique authorization token that is trusted by other OpenStack services. The token is valid for a limited period of time. With the authorization token, users can use a service's REST API to request other service tasks.

A token is a unique code with user roles encapsulated. When a user wants to perform a task in Red Hat OpenStack Platform, the task service requests the user's token. If the token does not exist or is expired, the user obtains a token from the identity service by authenticating. A personal token is then generated and received. The task service validates the token and checks the contained privileges. The client then performs the requested task, or the request is rejected due to insufficient token privileges. The **openstack token issue** command can be used to generate a new token.

UNIFIED CLI

The unified CLI is an easy way to interact with Red Hat OpenStack Platform services. The **openstack** command supports many commands to interact with the other Red Hat OpenStack Platform services, perform various actions, and provide options and arguments to customize the result and output. The **openstack** command uses the following syntax:

```
[user@demo ~(user-demo)]$ openstack global-options command command-
action arguments
```

Example Unified CLI Subcommands

COMMAND	DESCRIPTION
user	Creating and managing users
project	Creating and managing projects and quotas
server	Deploying and managing server instances
flavor	Create and managing deployment sizing flavors

COMMAND	DESCRIPTION
image	Creating and managing template images and image volumes
volume	Creating and managing block-based storage as volumes
network	Creating and managing networking resources and connections



NOTE

Command names may consist of multiple words. For instance, the **openstack floating ip** command manages floating IP addresses.

Sample Unified CLI Command Actions

OPTION	DESCRIPTION
create	Create a new resource
delete	Delete an existing resource
list	List existing resources
show	Examine a single existing resource's details
set	Change or add additional attribute parameters to an existing resource
unset	Remove additional attribute parameters from an existing resource

For example, the **openstack flavor list** command can be used to return basic parameters for available flavors.

```
[user@demo ~(user-demo)]$ openstack flavor list
+-----+-----+-----+-----+-----+-----+
| ID      | Name     | RAM | Disk | Ephemeral | vCPUs | Is Public |
+-----+-----+-----+-----+-----+-----+
| 6586...fc13 | default | 512 | 10 | 0 | 1 | True |
+-----+-----+-----+-----+-----+-----+
```

This flavor is listed as *Public*, meaning it can be seen and used by all projects and users.

To display the details of a specific flavor, use the **openstack flavor show** command with the flavor name as an argument. This command displays the details of the specified flavor, including its disk size, amount of RAM, swap size, ephemeral disk, vCPUs, and status.

```
[user@demo ~(user-demo)]$ openstack flavor show default
+-----+-----+
| Field          | Value           |
+-----+-----+
| OS-FLV-DISABLED:disabled | False          |
| OS-FLV-EXT-DATA:ephemeral | 0              |
| access_project_ids | None           |
| disk            | 10             |
```

```

| id           | 6586868d-c5bb-4e09-b68f-392a755afc13 |
| name         | default                                |
| os-flavor-access:is_public | True                                 |
| properties   |                                         |
| ram          | 2048                                  |
| rxtx_factor  | 1.0                                   |
| swap          |                                         |
| vcpus         | 2                                     |
+-----+-----+

```

Use the **openstack project show** command with a project name to display details about one project. Users with admin privileges can use this command to examine any available project.

```

[user@demo ~] $ openstack project show demo
+-----+-----+
| Field      | Value           |
+-----+-----+
| description | demo            |
| domain_id   | default          |
| enabled     | True             |
| id          | 903ebb6e211a4bf093f8a7215784682e |
| is_domain   | False            |
| name        | demo             |
| parent_id   | default          |
| tags        | []               |
+-----+-----+

```

The output of this command displays the project name, the project ID, and the status of the project. In that example, the project is enabled, meaning that the resources in that project can be used. For example, a user can launch a server instance using the project's resources.

Use the **openstack user show** command with a user name to display details about a one user account. This command displays the user ID, their name, their primary project ID, and the user's status. In the following example, the user is enabled, which means that this account is active and can be used to perform tasks in Red Hat OpenStack Platform.

```

[user@demo ~] $ openstack user show user
+-----+-----+
| Field      | Value           |
+-----+-----+
| default_project_id | 903ebb6e211a4bf093f8a7215784682e |
| domain_id   | default          |
| email       | user@lab.example.com |
| enabled     | True             |
| id          | fc318aa670314cc08148c4828b71bdc2 |
| name        | user             |
| options     | {}               |
| password_expires_at | None            |
+-----+-----+

```

Use the **openstack image list** command to view available images that this user can use. For each available image listed, the name of the image, its ID, and information regarding its status is displayed. In the following example, the status is **active**, indicating the image can be used to launch an instance.

```
[user@demo ~(user-demo)]$ openstack image list
+-----+-----+-----+
| ID | Name | Status |
+-----+-----+-----+
| 6533cafe-7629-48ea-aef1-1efcc7a09913 | rhe17 | active |
+-----+-----+-----+
```

The **openstack server list** command displays a list of instances. For each instance, the output includes the instance name, ID, current status, IP address(es), the name of the network the instance is deployed on, and the image from which the instance was originally booted. In the following example, the **active** status indicates the instance is running.

```
[user@demo ~(user-demo)]$ openstack server list
+-----+-----+-----+-----+-----+
+-----+-----+-----+-----+-----+
| ID | Name | Status | Networks | Image |
| Flavor | | | | |
+-----+-----+-----+-----+-----+
+-----+
| 273e...4607 | demo-server1 | ACTIVE | demo-network1=192.168.1.X | rhe17 |
| default | |
+-----+-----+-----+-----+-----+
+-----+
```

The **openstack server show** command displays additional details about a specific instance. The instance name is passed as an argument. This command's output includes the instance name, ID, current status, IP address(es), the name of the network the instance is deployed on, the image from which the instance was originally booted, the creation date, and the hypervisor on which his instance is currently running. The fields that are named beginning with the text "OS-" can only be displayed by users with admin privileges.

```
[user@demo ~(user-demo)]$ openstack server show web-server
+-----+-----+
| Field | Value |
+-----+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2018-06-01T07:45:26.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4 | |
| accessIPv6 | |
| addresses | demo-network1=192.168.1.X |
| config_drive | |
| created | 2018-06-01T07:45:07Z |
| flavor | default (6586868d-c5bb-4e09-b68f-392a755afc13) |
| hostId | 2e58...f2c0 |
| id | 273edb27-b4aa-46ac-b820-238b8b7c4607 |
| image | rhe17 (6533cafe-7629-48ea-aef1-1efcc7a09913) |
| key_name | None |
| name | demo-server1 |
| progress | 0 |
```

```
| project_id          | 903ebb6e211a4bf093f8a7215784682e
| properties          |
| security_groups    | name='default'
| status              | ACTIVE
| updated             | 2018-06-01T07:45:26Z
| user_id             | fc318aa670314cc08148c4828b71bdc2
| volumes_attached   |
+-----+
```

Use the **openstack help** command to get additional information supported command usage.

The following command lists available commands for the **openstack** CLI.

```
[user@demo ~(user-demo)]$ openstack help
```

The following command lists the available command actions for any specified command.

```
[user@demo ~(user-demo)]$ openstack help command
```

The following command lists available arguments for a specified command and action.

```
[user@demo ~(user-demo)]$ openstack help command command-action
```

FORMATTING UNIFIED CLI OUTPUT

The OpenStack CLI command default output is printed using a tabular format. A different machine parseable output can be specified with the **--format** or **-f** option to **list** and **show** commands. Commonly supported output formats include JSON, CSV, text, or shell styled outputs. In the following example, **openstack server list** is displayed using JSON structure:

```
[user@demo ~(user-demo)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "demo-server1",
    "Image": "rhel7",
    "ID": "273edb27-b4aa-46ac-b820-238b8b7c4607",
    "Flavor": "default",
    "Networks": "demo-network1=192.168.1.X"
  }
]
```

The **show** commands have an option **-f shell** for storing the output values in shell variables.

```
[user@demo ~(user-demo)]$ openstack server show \
-f shell --prefix my_ demo-server1
...output omitted...
my_addresses="demo-network1=192.168.1.X"
my_config_drive=""
my_created="2018-06-01T07:45:26.000000"
my_flavor="default (6586868d-c5bb-4e09-b68f-392a755afc13)"
my_hostid="2e58...f2c0"
my_id="273edb27-b4aa-46ac-b820-238b8b7c4607"
```

```
my_image="rhel7 (6533cafe-7629-48ea-aef1-1efcc7a09913)"  
my_key_name="None"  
my_name="demo-server1"  
...output omitted...
```

DISPLAYING DEBUG INFORMATION

Using incorrect parameters or conflicting resource names in OpenStack CLI can result in displayed errors. The default logging level is set to INFO, which prints limited debugging information. For more debugging output, use the **--debug** option to set the logging level to DEBUG. The log output is displayed on the standard error device by default. Use the **--log-file** option to send the logging output to a specified file.

The example below shows DEBUG level information displayed using the **--debug** option. The user mistakenly used an incorrect flavor name to create an instance.

```
[user@demo ~(user-demo)]$ openstack server create \  
--flavor nodefault --image rhel7 --debug demo-server1  
...output omitted...  
CommandError: No flavor with a name or ID of 'nodefault' exists.  
  
END return value: 1
```

Using the **openstack** Command

- Use the **openstack help** command to retrieve information about what commands are available.
- Use the **openstack help command** command to display information about what command actions are available.
- Use the **openstack help command command-action** command to get information about a specific command and actions.



NOTE

The **openstack complete** command provides a bash-completion script for the **openstack** command. The following command configures this bash-completion script system-wide.

```
[user@demo ~]$ openstack complete | sudo tee /etc/bash_completion.d/openstack  
> /dev/null
```



REFERENCES

Further information is available in the *Integrate with Identity Service* guide for Red Hat OpenStack Platform at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/integrate_with_identity_service/pr01

Further information is available in the OpenStack command-line clients section of the *User Guide* for OpenStack at <https://docs.openstack.org/user-guide/>

RUNNING THE OPENSTACK UNIFIED CLI

In this exercise, you will use the unified command-line interface (CLI) to list and check the details of existing projects, users, flavors, images, and instances.

OUTCOMES

You should be able to describe the Identity service environment file and use the unified command-line interface.

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

On **workstation**, run **lab instances-cli setup**, which ensures that the **developer1-finance-rc** file exists. The script also creates the **finance-server1** instance.

```
[student@workstation ~]$ lab instances-cli setup
```

- ▶ 1. Open a terminal and check the **developer1** credentials included in the **developer1-finance-rc** environment file in the **student** home directory. The user name for this exercise is **developer1**, and the password is **redhat**. The user belongs to the project **finance**. The IP address of the identity service endpoint corresponds to a load-balanced virtual IP address for the **controller0** overcloud node.

```
[student@workstation ~]$ cat developer1-finance-rc
...output omitted...
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_PASSWORD=redhat
export OS_PROJECT_DOMAIN_NAME=Default
export OS_PROJECT_NAME=finance
export OS_USERNAME=developer1
export OS_USER_DOMAIN_NAME=Default
...output omitted...
```

- ▶ 2. Source the **developer1-finance-rc** file to gain the appropriate user privileges in the Red Hat OpenStack Platform environment. This enables all the **OS_*** environment variables containing, at a minimum, the **developer1** credentials and the Identity service public endpoint.

```
[student@workstation ~]$ source developer1-finance-rc
[student@workstation ~(developer1-finance)]$
```

- ▶ 3. Verify that the **OS_*** environment variables have been exported to the shell environment. The **OS_USERNAME** should be set to **developer1**, **OS_PASSWORD** should be set to **redhat**, **OS_PROJECT_NAME** should be set to **finance**, and **OS_AUTH_URL** should be set to **http://172.25.250.50:5000/v3**, which represents the public IP address for the

controller0 virtual machine hosting the identity service, the public access endpoint port 5000, and the Identity service API version 3.0.

```
[student@workstation ~ (developer1-finance)]$ env | grep OS_
...output omitted...
OS_PASSWORD=redhat
OS_AUTH_URL=http://172.25.250.50:5000/v3
OS_USERNAME=developer1
OS_PROJECT_NAME=finance
```

- 4. Use the **openstack help project show** command to determine how to display details of a particular project.

```
[student@workstation ~ (developer1-finance)]$ openstack help project show
usage: openstack project show [-h] [-f {json,shell,table,value,yaml}]
                               [-c COLUMN] [--max-width <integer>]
                               [--fit-width] [--print-empty] [--noindent]
                               [--prefix PREFIX] [--domain <domain>]
                               [--parents] [--children]
                               <project>
```

Display project details

positional arguments:
 <project> Project to display (name or ID)
...output omitted...

- 5. Gather additional information about the **developer1** user's current project, named **finance**.

```
[student@workstation ~ (developer1-finance)]$ openstack project show finance
+-----+-----+
| Field      | Value
+-----+-----+
| description | finance
| domain_id   | default
| enabled     | True
| id          | 903ebb6e211a4bf093f8a7215784682e
| is_domain   | False
| name        | finance
| parent_id   | default
| tags        | []
+-----+-----+
```

- 6. Use the **openstack help user show** command to determine how to display details of a specific user account.

```
[student@workstation ~ (developer1-finance)]$ openstack help user show
usage: openstack user show [-h] [-f {json,shell,table,value,yaml}] [-c COLUMN]
                           [--max-width <integer>] [--fit-width]
                           [--print-empty] [--noindent] [--prefix PREFIX]
                           [--domain <domain>]
                           <user>
```

```
Display user details

positional arguments:
  <user>                  User to display (name or ID)
...output omitted...
```

- 7. List the details of the **developer1** account.

```
[student@workstation ~ (developer1-finance)]$ openstack user show developer1
+-----+-----+
| Field      | Value   |
+-----+-----+
| default_project_id | 903ebb6e211a4bf093f8a7215784682e |
| domain_id    | default |
| email        | developer1@lab.example.com |
| enabled       | True    |
| id           | fc318aa670314cc08148c4828b71bdc2 |
| name         | developer1 |
| options       | {}      |
| password_expires_at | None   |
+-----+-----+
```

- 8. Use the **openstack help flavor list** command to determine how to display all available flavors.

```
[student@workstation ~ (developer1-finance)]$ openstack help flavor list
usage: openstack flavor list [-h] [-f {csv,json,table,value,yaml}] [-c COLUMN]
                               [--max-width <integer>] [--fit-width]
                               [--print-empty] [--noindent]
                               [--quote {all,minimal,none, numeric}]
                               [--sort-column SORT_COLUMN]
                               [--public | --private | --all] [--long]
                               [--marker <flavor-id>] [--limit <num-flavors>]

List flavors
...output omitted...
```

- 9. Use the **openstack flavor list** command to list all available flavors.

```
[student@workstation ~ (developer1-finance)]$ openstack flavor list
+-----+-----+-----+-----+-----+-----+
| ID      | Name     | RAM | Disk | Ephemeral | VCPUs | Is Public |
+-----+-----+-----+-----+-----+-----+
| 6586...fc13 | default | 2048 | 10 | 0 | 2 | True |
+-----+-----+-----+-----+-----+-----+
```

- 10. Use the **openstack flavor show default** command to display details of the **default** flavor.

```
[student@workstation ~ (developer1-finance)]$ openstack flavor show default
+-----+-----+
```

Field	Value
OS-FLV-DISABLED:disabled	False
OS-FLV-EXT-DATA:ephemeral	0
access_project_ids	None
disk	10
id	6586868d-c5bb-4e09-b68f-392a755afc13
name	default
os-flavor-access:is_public	True
properties	
ram	2048
rxtx_factor	1.0
swap	
vcpus	2

- 11. Use the **openstack image list** command to list all available images.

```
[student@workstation ~ (developer1-finance)]$ openstack image list
+-----+-----+-----+
| ID      | Name   | Status |
+-----+-----+-----+
| 6533cafe-7629-48ea-aef1-1efcc7a09913 | rhel7  | active |
+-----+-----+-----+
```

- 12. Use the **openstack help server** command to determine how to list all instances.

```
[student@workstation ~ (developer1-finance)]$ openstack help server
Command "server" matches:
  server add fixed ip
  server add floating ip
  server add security group
  server add volume
  server backup create
  server create
  server delete
  server dump create
  server group create
  server group delete
  server group list
  server group show
  server image create
  server list
  server lock
  server migrate
  server pause
  server reboot
...output omitted...
```

- 13. Use the **openstack server list -f json** command to list all available instances.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
```

```

    {
        "Status": "ACTIVE",
        "Name": "finance-server1",
        "Image": "rhel7",
        "ID": "d7fe0d57-2490-40f9-a5b4-0945332d1610",
        "Flavor": "default",
        "Networks": "finance-network1=192.168.1.X"
    }
]

```

- 14. Use the **openstack server show finance-server1** command to display details of the **finance-server1** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server show finance-server1
+-----+-----+
| Field          | Value
+-----+-----+
| OS-DCF:diskConfig | MANUAL
| OS-EXT-AZ:availability_zone | nova
| OS-EXT-STS:power_state | Running
| OS-EXT-STS:task_state | None
| OS-EXT-STS:vm_state | active
| OS-SRV-USG:launched_at | 2018-06-01T07:45:26.000000
| OS-SRV-USG:terminated_at | None
| accessIPv4 |
| accessIPv6 |
| addresses | finance-network1=192.168.1.X
| config_drive |
| created | 2018-06-01T07:45:07Z
| flavor | default (6586868d-c5bb-4e09-b68f-392a755afc13)
| hostId | 2e58...f2c0
| id | d7fe0d57-2490-40f9-a5b4-0945332d1610
| image | rhel7 (6533cafe-7629-48ea-aef1-1efcc7a09913)
| key_name | None
| name | finance-server1
| progress | 0
| project_id | 903ebb6e211a4bf093f8a7215784682e
| properties |
| security_groups | name='default'
| status | ACTIVE
| updated | 2018-06-01T07:45:26Z
| user_id | fc318aa670314cc08148c4828b71bdc2
| volumes_attached |
+-----+-----+
```

Cleanup

On **workstation**, run the **lab instances-cli** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab instances-cli cleanup
```

This concludes the guided exercise.

LAUNCHING AN INSTANCE USING THE CLI

OBJECTIVE

After completing this section, students should be able to launch an instance using the command-line interface.

LAUNCHING AN INSTANCE USING THE COMMAND-LINE INTERFACE

Instances are deployed by users within a project. In a default project environment, instances can only connect to other instances in the same project. Public network access can be configured through a project router to connect with external networks. Instances can be given access to other networks and public resources through an external network.

A flavor is a set of runtime resources, such as disk drives and memory, that are assigned to an instance when it is deployed. To deploy an instance, an existing flavor, image and network must be available and specified.

NOTE

To use an image with a particular flavor, the flavor must meet or exceed the image's minimum configured resource requirements. For example, the flavor must be configured to satisfy an image's minimum disk space and RAM settings and needs.

One characteristic that distinguishes cloud environments from legacy servers is that installing operating systems or software packages in online virtual instances is inefficient. Deployed cloud instances use prebuilt images, similar to the templates used in enterprise virtualization. Disk images are created and used in the Red Hat OpenStack Platform environment to provide a preinstalled operating system with preconfigured services, added software and common configuration. Cloud administrators prepare different images (for example: web server, ftp server, and database server images) that support the organization's business needs in the data center. After an image is created and uploaded, it can be used to create an instance.

The **openstack server image create** command can also be used to create images. This command creates a new image from the system disk of a running instance, storing it in the image service image store. An OpenStack operator could deploy a minimal base image with the desired operating system version, then add additional features, packages and customizations to the running system. After appropriate testing and validation, the running instance is saved as a new image, which can then be used to deploy any quantity of identical instances.

A newly created instance is assigned to be protected by one or more *security groups*. A security group is a named collection of netfilter access rules that limit the traffic types that can access an instance. New instances are automatically assigned the **default** security group, unless specified otherwise. The rules in each security group actively control traffic to instances; the rules can be modified in real time to add or remove traffic protocols without needing to redeploy the server instance.

The **openstack server** command includes multiple command actions. Use the **openstack help server** command to display the complete list of command actions:

```
[user@demo ~(admin-admin)]$ openstack help server
...output omitted...
server add fixed ip
server add floating ip
server add security group
server add volume
server backup create
server create
server delete
server dump create
server group create
server group delete
server group list
server group show
server image create
server list
server lock
server migrate
server pause
server reboot
server rebuild
server remove fixed ip
server remove floating ip
server remove security group
...output omitted...
```

This chapter focuses primarily on the **create** and **set** command actions. The **create** command action creates resources needed to launch a new instance. The **set** command action adds or changes attribute properties on existing instance resources.

```
[user@demo ~(admin-admin)]$ openstack help server set
usage: openstack server set [-h] [--name new-name] [--root-password]
                           [--property key=value] [--state state]
                           server

Set server properties

positional arguments:
  server           Server (name or ID)

optional arguments:
  -h, --help        show this help message and exit
  --name new-name   New server name
  --root-password  Set new root password (interactive only)
  --property key=value Property to add/change for this server (repeat option
                        to set multiple properties)
  --state state     New server state (valid value: active, error)
```



NOTE

In cloud terminology, **instance** and **server** are both used to describe a cloud image-deployed virtual machine. From an end user's viewpoint, deployed cloud applications behave like servers. From an administrator's viewpoint, a logical server might be a server farm comprised of dozens of load-balanced instances.

Complex server applications may be a combination of diverse application instances configured to work together to appear as a single service. This advanced deployment concept, known as a **stack**, is covered in a later chapter.

In this coursebook, **instance** and **server** are used interchangeably, matching common documentation usage. The **openstack** command uses the word **server**.

One example of using **openstack server set** is for interactively changing the root password inside an active instance.

```
[user@demo ~(user-demo)]$ openstack server set --root-password demo-server1  
New password:redhat  
Retype new password:redhat
```



NOTE

To successfully use the previous command to change the **root** password on a running instance, the instance must be running the **qemu-guest-agent** service. This service requires that the host hypervisor machine is connected to the guest VM machine through a VirtIO serial connection.

See How to enable QEMU guest agent in KVM [<https://access.redhat.com/solutions/732773>].

Launching an Instance with the CLI

The **create** command action is used with the **openstack server** command to launch a new instance from the CLI.

```
[user@demo ~(user-demo)]$ openstack server create --flavor default --image rhel7  
demo-server1  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| OS-DCF:diskConfig | MANUAL |  
| OS-EXT-AZ:availability_zone | |  
| OS-EXT-SRV-ATTR:host | None |  
| OS-EXT-SRV-ATTR:hypervisor_hostname | None |  
| OS-EXT-SRV-ATTR:instance_name | |  
| OS-EXT-STS:power_state | NOSTATE |  
| OS-EXT-STS:task_state | scheduling |  
| OS-EXT-STS:vm_state | building |  
| OS-SRV-USG:launched_at | None |  
| OS-SRV-USG:terminated_at | None |  
| accessIPv4 | |  
| accessIPv6 | |  
| addresses | |
```

config_drive	
created	2018-06-01T10:55:59Z
flavor	default (6586...fc13)
hostId	
id	8f4475bb-6a34-4ad2-bc68-cc63487ae3fb
image	rhel7 (653f...9913)
key_name	None
name	demo-server1
progress	0
project_id	903ebb6e211a4bf093f8a7215784682e
properties	
security_groups	name='default'
status	BUILD
updated	2018-06-01T10:55:59Z
user_id	683629e2f20041bea22ffac7e40bf632
volumes_attached	

When the deployment of an instance is started, the Red Hat OpenStack Platform compute service searches for a compute node with sufficient resources to deploy the instance. The flavor specifies the resources required by the instance. After the compute node has been selected, the base image is transferred to this node, expanding it to the requested root disk size. When the transfer and resizing are complete, the instance is booted.

Instances are deployed using the **default** security group unless another is specified. Unmodified, the **default** security group allows any outgoing traffic from the instance to anywhere, and any incoming traffic but only from other instances deployed using the same **default** security group. To enable incoming traffic from other sources not configured with the same security group, explicit rules must be added to the default security group for only those protocols to be needed from other sources. Alternatively, create and customize a new security group to meet the explicit requirements of the server's applications instead of changing the project's default security group.

Listing, Stopping, Starting, and Deleting Instances with the CLI

When instances are up and running, use the **openstack** command to perform tasks on available instances. The **openstack server list** command displays all the available instances.

[user@demo ~(user-demo)]\$ openstack server list				
ID	Name	Status	Networks	Image Name
ac0dd(...) 78d3261c362	demo1	ERROR		demo-img
7051e(...) f50418b4a79	demo2	ACTIVE	lab=192.168.0.10	demo-img

The previous example indicates there are two instances, one active and one with error status. Both instances were created with the same image. The active instance has a network associated with it and an IP address allocated.

The instance **Name** or **ID** is required to start or delete an instance. Recommended practice, especially in scripting, is to use the **ID** instead of the **Name**, because OpenStack allows the creation of multiple instances with the same name. This practice is common in resilient, highly-available application design.

The **openstack server stop** command stops a running instance while leaving the instance resident in memory. The instance name or ID is specified as the argument to the command.

```
[user@demo ~(user-demo)]$ openstack server stop instance-name
```

The **openstack server show** command displays the details about an existing instance. This command displays the IP address and run status of the instance, and other details such as the flavor, image, and project used by the the instance.

The following example views the details for the "Error" state instance from the previous display.

```
[user@demo ~(user-demo)]$ openstack server show demo1
+-----+-----+
| Field | Value |
+-----+-----+
| OS-DCF:diskConfig | MANUAL
| OS-EXT-AZ:availability_zone |
| OS-EXT-STS:power_state | NOSTATE
| OS-EXT-STS:task_state | None
| OS-EXT-STS:vm_state | error
| OS-SRV-USG:launched_at | None
| OS-SRV-USG:terminated_at | None
| accessIPv4 |
| accessIPv6 |
| addresses |
| config_drive |
| created | 2017-03-07T11:44:21Z
| fault | {u'message': u"Build of instance
| ac0ddb44-996f-4710-84c0-a78d3261c362
| aborted: Flavor's disk is too small
| for requested image. Flavor disk is
| 1073741824 bytes, image is
| 10737418240 bytes.", u'code': 500,
| u'created': u'2017-03-07T11:44:29Z'}
| flavor | demo.flavor (1)
| hostId |
| id | ac0ddb44-996f-4710-84c0-a78d3261c362
| image | demo-img (a345777f-ab75-4fbf-
| b7a0-dc8045912e93)
| key_name | None
| name | demo1
| os-extended-volumes:volumes_attached | []
| project_id | facb05290070461da5f48616ed6e736b
| properties |
| status | ERROR
| updated | 2017-03-07T11:44:29Z
| user_id | c65470ff6e964de9a7867971b884fe58
+-----+-----+
```

The **show** subcommand displays the reason why the instance did not launch. In the example, the image used was larger than the flavor allowed, causing the error.

Use the same command on a running instance to view information about resources used during creation of the instance:

```
[user@demo ~(user-demo)]$ openstack server show demo2
+-----+-----+
```

Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2017-03-07T10:22:40.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	lab=192.168.0.10
config_drive	
created	2017-03-07T10:22:17Z
flavor	demo-flavor (42)
hostId	c5850f3f3c74185f1c10c92e4802aa4e96f5c
	d9135406a0c6bd281b8
id	7051e207-99b9-4652-808c-7f50418b4a79
image	demo-img (a345777f-ab75-4fbf- b7a0-dc8045912e93)
key_name	None
name	demo2
os-extended-volumes:volumes_attached	[]
progress	0
project_id	facb05290070461da5f48616ed6e736b
properties	
security_groups	[{"name": "default"}]
status	ACTIVE
updated	2017-03-07T12:03:42Z
user_id	c65470ff6e964de9a7867971b884fe58

It is not required to stop an instance before removing it from the Red Hat OpenStack Platform environment. Use the **openstack server delete** command with the instance name or ID to delete a running or stopped instance.

```
[user@demo ~]$(user-demo)]$ openstack server delete server-name
```

Connecting to an Instance with the CLI

While an instance is running, use the CLI to examine the instance's log files, or to display the URL used to access the instance's console. The **openstack console log show** command displays the current log entries of an instance.

```
[user@demo ~]$(user-demo)]$ openstack console log show demo-server1
... output omitted ...
[[32m OK [0m] Started Network Manager Script Dispatcher Service.
Starting Hostname Service...
[ 37.327203] IPv6: ADDRCONF(NETDEV_UP): eth0: link is not ready
[[32m OK [0m] Started Hostname Service.
[ 39.425321] cloud-init[497]: Cloud-init v. 0.7.6 running 'init-local' at Tue,
07 Mar 2017 12:05:05 +0000. Up 39.36 seconds.
```

Red Hat Enterprise Linux Server 7.3 (Maipo)
Kernel 3.10.0-514.6.1.el7.x86_64 on an x86_64

```
web image
host-192-168-1-10 login: [ 52.312776] cloud-init[796]: Cloud-init v. 0.7.6 run
ning 'init' at Tue, 07 Mar 2017 12:05:18 +0000. Up 52.21 seconds.
... output omitted ...
```

The **openstack console url show** command displays the instance's VNC console URL. A cloud user can log in to the instance using a browser, even when an instance has no connected network to allow SSH access.

```
[user@demo ~(user-demo)]$ openstack console url show demo-server1
+-----+
| Field | Value
+-----+
| type  | novnc
| url   | http://172.25.250.50:6080/vnc_auto.html?token=6f1f90d7-97ac-
|       | 4d98-bd16-1762cefb305a
+-----+
```

Navigate to the displayed URL in a browser. It connects to the VNC console directly, without requiring the OpenStack dashboard.

Launching an Instance Using the CLI

The following steps outline the process of launching an instance and listing available resources using the unified CLI.

1. Source the appropriate Identity service credentials environment file.
2. View the details of the available resources (for example flavors, images, and networks).
3. List the available instances.
4. Launch a new instance from the CLI.
5. Launch another instance with the same name.
6. List the current instances. Use the instance name to further inspect instances.
7. Stop the first instance.
8. Delete the first instance.
9. Restart the running instance.
10. Access the instance console log.
11. Access the instance console.
12. Delete the remaining instance.



REFERENCES

Further information is available in the Virtual Machine Instances section of the *Instances and Images Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/instances_and_images_guide/

Further information is available in the Enhancing Virtualization with the Qemu Guest Agent and Spice Agent section of the *Virtualization Deployment and Administration Guide* for Red Hat Enterprise Linux at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/virtualization_deployment_and_administration_guide/chap-qemu_guest_agent/

LAUNCHING AN INSTANCE USING THE CLI

In this exercise, you will launch an instance using the command-line interface.

OUTCOMES

You should be able to:

- Verify all the resources needed for an instance to launch.
- Launch a new instance.
- Verify that the new instance uses the correct image and flavor.
- Delete an instance using its ID.
- Stop, start, and delete an instance using its name.
- Connect to the instance's console.
- Ping the DHCP server from the instance.

BEFORE YOU BEGIN

Ensure that **workstation** and the overcloud's virtual machines are started.

If not already logged in, log in to **workstation** as **student** using the password of **student**.

From **workstation**, run the **lab launch-cli** script with the **setup** argument, which verifies or creates the required resources for this exercise, including users, passwords, images, and networks for launching an instance.

```
[student@workstation ~]$ lab launch-cli setup
```

- ▶ 1. Source the **developer1** environment file, **/home/student/developer1-finance-rc**, from the command line. The environment file sets the identity service authentication endpoint, and the user's name, password, domain, and project to be referenced by the OpenStack unified CLI.

```
[student@workstation ~]$ source ~/developer1-finance-rc  
[student@workstation ~(developer1-finance)]$
```

- ▶ 2. Use the **openstack flavor list** command to list all the available flavors.

```
[student@workstation ~(developer1-finance)]$ openstack flavor list  
+-----+-----+-----+-----+-----+-----+  
| ID      | Name     | RAM | Disk | Ephemeral | VCPUs | Is Public |  
+-----+-----+-----+-----+-----+-----+
```

```
| 6586...fc13 | default | 2048 | 10 | 0 | 2 | True |
```

- 3. Use the **openstack image list** command to list all the available images.

```
[student@workstation ~ (developer1-finance)]$ openstack image list
+-----+-----+-----+
| ID | Name | Status |
+-----+-----+-----+
| 6533cafe-7629-48ea-aef1-1efcc7a09913 | rhel7 | active |
+-----+-----+-----+
```

- 4. Use the **openstack image show** command with the name of the image to see detailed information about the image.

```
[student@workstation ~ (developer1-finance)]$ openstack image show rhel7
+-----+
| Field | Value |
+-----+
| checksum | c4fdd1059af61b68ce342a6ed35cb357 |
| container_format | bare |
| created_at | 2018-06-01T07:38:17Z |
| disk_format | qcow2 |
| file | /v2/images/6533cafe-7629-48ea-aef1-1efcc7a09913/file |
| id | 6533cafe-7629-48ea-aef1-1efcc7a09913 |
| min_disk | 10 |
| min_ram | 2048 |
| name | rhel7 |
| owner | 903ebb6e211a4bf093f8a7215784682e |
| properties | direct_url='rbd://6e503d0c-635a-11e8-b2fc-52540001fac8/images/6533cafe-7629-48ea-aef1-1efcc7a09913/snap', locations='[{u'url': u'rbd://6e503d0c-635a-11e8-b2fc-52540001fac8/images/6533cafe-7629-48ea-aef1-1efcc7a09913/snap', u'metadata': {}}]' |
| protected | True |
| schema | /v2/schemas/image |
| size | 844890112 |
| status | active |
| tags | |
| updated_at | 2018-06-01T07:38:17Z |
| virtual_size | None |
| visibility | public |
+-----+
```

- 5. Use the **openstack server list** command to list all the available instances. The IP address for the instance might differ from the output.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server1",
```

```

        "Image": "rhel7",
        "ID": "62e92f09-e8e1-4e95-a555-00e8bac4fe94",
        "Flavor": "default",
        "Networks": "finance-network1=192.168.1.X"
    }
]

```

- 6. Create a new instance with the same name as the existing one. Use the **openstack server create** command to create another **finance-server1** instance. Use the **rhel7** image, **finance-network1** as the network and **default** as the flavor.

```
[student@workstation ~ (developer1-finance)]$ openstack server create --image rhel7
 \
--flavor default --nic net-id=finance-network1 --wait finance-server1
+-----+
| Field           | Value
+-----+
| OS-DCF:diskConfig | MANUAL
| OS-EXT-AZ:availability_zone | nova
| OS-EXT-STS:power_state | Running
| OS-EXT-STS:task_state | None
| OS-EXT-STS:vm_state | active
| OS-SRV-USG:launched_at | 2018-06-01T11:55:51.000000
| OS-SRV-USG:terminated_at | None
| accessIPv4 |
| accessIPv6 |
| addresses | finance-network1=192.168.1.Y
| adminPass | FdP6Zc9Hz7KS
| config_drive |
| created | 2018-06-01T11:55:39Z
| flavor | default (6586868d-c5bb-4e09-b68f-392a755afc13)
| hostId | f0df...635d
| id | 89f48040-970c-434f-bb0f-6fc5abd6b2fb
| image | rhel7 (6533cafe-7629-48ea-aef1-1efcc7a09913)
| key_name | None
| name | finance-server1
| progress | 0
| project_id | 903ebb6e211a4bf093f8a7215784682e
| properties |
| security_groups | name='default'
| status | ACTIVE
| updated | 2018-06-01T11:55:51Z
| user_id | fc318aa670314cc08148c4828b71bdc2
| volumes_attached |
+-----+
```

- 7. Use the **openstack server list** command to list all the available instances again.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
{
    "Status": "ACTIVE",
    "Name": "finance-server1",
    "Image": "rhel7",
```

```
"ID": "62e92f09-e8e1-4e95-a555-00e8bac4fe94",
"Flavor": "default",
"Networks": "finance-network1=192.168.1.X"
},
{
  "Status": "ACTIVE",
  "Name": "finance-server1",
  "Image": "rhel7",
  "ID": "89f48040-970c-434f-bb0f-6fc5abd6b2fb",
  "Flavor": "default",
  "Networks": "finance-network1=192.168.1.Y"
}
]
```



NOTE

Two instances now exist with the same name. In simple practice, this is not recommended, but it is common real-world application design. Therefore, specifying instances by name may not be explicit enough for OpenStack to understand to which instance a command refers. However, referencing an instance by its ID is always a unique reference.

- 8. Note the ID of the previously running instance to be able to stop it. Use the **openstack server stop** command with that instance ID.

```
[student@workstation ~ (developer1-finance)]$ openstack server stop \
62e92f09-e8e1-4e95-a555-00e8bac4fe94
```

- 9. Use the **openstack server list** command to again list all available instances.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "SHUTOFF",
    "Name": "finance-server1",
    "Image": "rhel7",
    "ID": "62e92f09-e8e1-4e95-a555-00e8bac4fe94",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.X"
  },
  {
    "Status": "ACTIVE",
    "Name": "finance-server1",
    "Image": "rhel7",
    "ID": "89f48040-970c-434f-bb0f-6fc5abd6b2fb",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.Y"
  }
]
```

- 10. Use the **openstack server show** command with a UUID to verify that the new and active instance uses the correct flavor and image.

```
[student@workstation ~(developer1-finance)]$ openstack server show \
89f48040-970c-434f-bb0f-6fc5abd6b2fb
+-----+-----+
| Field | Value |
+-----+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2018-06-01T11:55:51.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4 | |
| accessIPv6 | |
| addresses | finance-network1=192.168.1.Y |
| config_drive | |
| created | 2018-06-01T11:55:39Z |
| flavor | default (6586868d-c5bb-4e09-b68f-392a755afc13) |
| hostId | f0df...635d |
| id | 89f48040-970c-434f-bb0f-6fc5abd6b2fb |
| image | rhel7 (6533cafe-7629-48ea-aef1-1efcc7a09913) |
| key_name | None |
| name | finance-server1 |
| progress | 0 |
| project_id | 903ebb6e211a4bf093f8a7215784682e |
| properties | |
| security_groups | name='default' |
| status | ACTIVE |
| updated | 2018-06-01T11:55:51Z |
| user_id | fc318aa670314cc08148c4828b71bdc2 |
| volumes_attached | |
+-----+-----+
```

- 11. Delete the existing instance named **finance-server1**. In the previous sample output, it was the first instance that was in the **SHUTOFF** state. Use the **openstack server delete** command with the UUID of the instance to delete the instance.

```
[student@workstation ~(developer1-finance)]$ openstack server delete \
62e92f09-e8e1-4e95-a555-00e8bac4fe94
```

- 12. Use the **openstack server list** command to list all the available instances again.

```
[student@workstation ~(developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server1",
    "Image": "rhel7",
    "ID": "89f48040-970c-434f-bb0f-6fc5abd6b2fb",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.Y"
```

```
}
```

```
]
```

- 13. Use the **openstack console log show** command to verify the log for the active instance.

```
[student@workstation ~ (developer1-finance)]$ openstack console log show \
finance-server1
...output omitted...
Red Hat Enterprise Linux Server 7.5 (Maipo)
Kernel 3.10.0-862.el7.x86_64 on an x86_64

small image
finance-server1 login:
```

- 14. Use the **openstack url show** command to display the instance's console URL. The URL given by the output of that command can be used to open a browser session using the VNC console to access the instances console.

```
[student@workstation ~ (developer1-finance)]$ openstack console url show \
finance-server1
+-----+
| Field | Value |
+-----+
| type  | novnc |
| url   | http://172.25.250.50:6080/vnc_auto.html?token=962dfd71-f047-43d3-89 |
|       | a5-13cb88261eb9 |
+-----+
```

- 15. To view the console, open the URL link from the previous step with Firefox.

- 16. To verify the instance configuration, log in to the **finance-server1** instance as **root** user with the password **redhat**.

```
small image
finance-server1 login: root
Password: redhat
[root@finance-server1 ~]#
```

- 17. Use the **ping** command to reach the IP address of the DHCP server from inside the instance.

```
[root@finance-server1 ~]# ping -c3 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.642 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.457 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.596 ms
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.457/0.565/0.642/0.078 ms
```

- 18. Return to a **workstation** terminal and stop the instance. Use the **openstack server stop** command with the name of the instance.

```
[student@workstation ~](developer1-finance)]$ openstack server stop finance-server1
```

- 19. Delete the existing instance **finance-server1**. Use the **openstack server delete** command with the instance name.

```
[student@workstation ~](developer1-finance)]$ openstack server delete finance-server1
```

Cleanup

On **workstation**, run the **lab launch-cli cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab launch-cli cleanup
```

This concludes the guided exercise.

DESCRIBING THE OPENSTACK ARCHITECTURE

OBJECTIVE

After completing this section, students should be able to describe the Red Hat OpenStack Platform architecture and use cases.

RED HAT OPENSTACK PLATFORM OVERVIEW

Red Hat OpenStack Platform is implemented as a collection of interacting services that control compute, storage, and networking resources. The following diagram provides a high-level overview of the OpenStack core services.

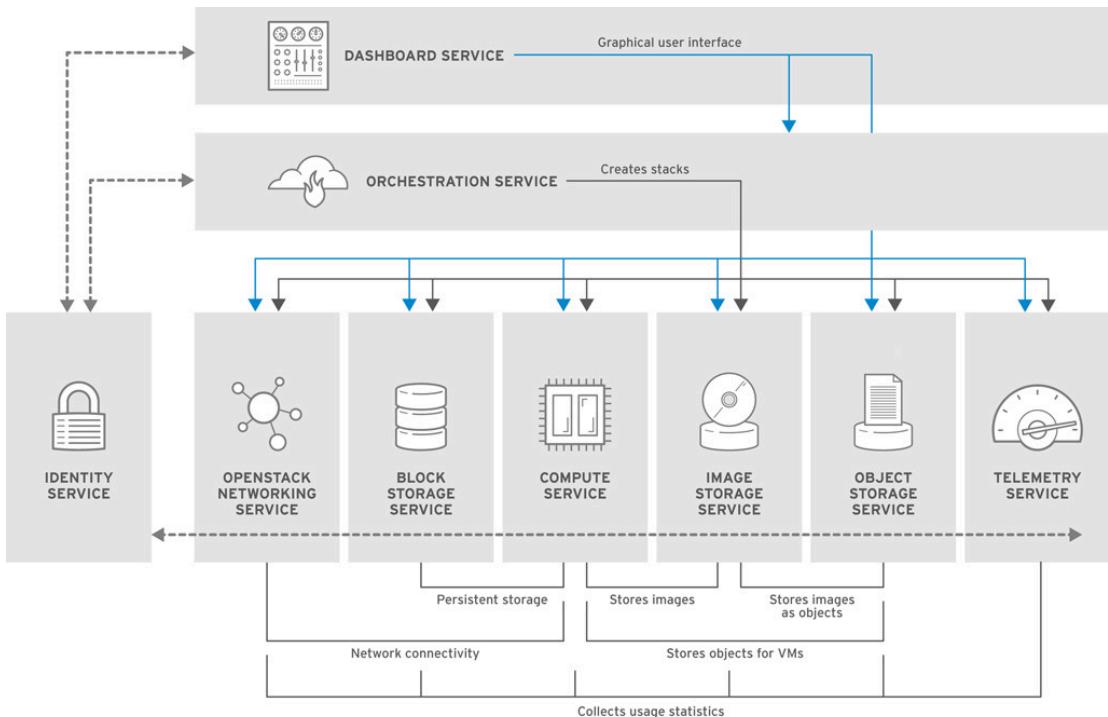


Figure 1.5: OpenStack core services

RHELOSP_347192_I015

SERVICES COVERED IN THIS COURSE

- Dashboard (horizon)

Dashboard is a browser-based interface for managing OpenStack services, providing a graphical interface for launching instances, managing networking, and setting access controls.

- Identity service (keystone)

The Identity service provides authentication and authorization to all OpenStack services. The service provides a central catalog of other services and associated endpoints, available in an OpenStack cloud. It supports multiple forms of authentication, including user name and password, token-based, and Amazon Web Services (AWS) logins. The Identity service acts as a single sign-on (SSO) authentication service for users and components. This service is responsible for creating and managing users, domains, roles, and projects.

- **Users:** The Identity service validates that requests are made by authenticated users. Identity uses tokens to provide user role information to access specific project resources.
- **Projects:** A project is a group of resources (e.g., users, images, instances, networks, volumes). The term *project* is equivalent to the older term *tenant* used in early versions of Red Hat OpenStack Platform. Projects assist to isolate and group identity objects and project resources. Depending on the desired implementation, a project can map to a customer, account, organization, or development environment.
- **Domains:** A domain defines the administrative boundaries of Identity service entities. A domain consists of one or more projects, users, and groups. In a multi-tenant cloud, multiple identity providers providing authentication and authorization service can be each associated with a separate domain. Resources configured in one domain are not shareable or accessible to other domains, by design. Additionally, resources may not be moved from one domain to another, they must be recreated in the new domain.

- **OpenStack Networking service (neutron)**

OpenStack Networking service is a software-defined networking (SDN) service that helps to create networks, subnets, routers, and floating IP addresses. Users can create and attach interface devices to instances and routers. Administrators can define a gateway for a router to allow external access. OpenStack networking ships with plug-ins and agents for Cisco virtual and physical switches, Open vSwitch, OVN, and others. The common agents are L3 and DHCP (which provides DHCP IP addresses to instances). OpenStack networking enables projects to create advanced virtual network topologies including entities such as firewalls, load balancers, and virtual private networks (VPNs).

- **Block Storage service (cinder)**

The Block Storage service manages storage volumes for virtual machines. This can be both ephemeral and persistent block storage for instances running in the Compute service. Snapshots can be taken for backing up data, either for restoring data or to be used to create new block storage volumes.

- **Compute service (nova)**

The Compute service manages instances (virtual machines) running on nodes, providing virtual machines on demand. It is a distributed service and interacts with the Identity service for authentication, Image service for images, and dashboard as a web-based user interface. The Compute service is designed to scale out horizontally on standard hardware, downloading images to launch instances as required. The Compute service uses **libvirt**, **qemu**, and **kvm** for the hypervisor.

- **Image service (glance)**

The Image service acts as a registry for virtual machine images, allowing users to copy server images for immediate storage. These images can be used as templates when setting up new instances.

- **Object Store service (swift)**

The Object Store provides object storage that allows users to store and retrieve files. The Object Store architecture is distributed to allow for horizontal scaling and to provide redundancy as a design to avoid failure or object loss.

One use case for Object Store is that it can act as a storage back end for images. This allows OpenStack to replicate images and snapshots across the Object Store infrastructure. This solution is useful as a backup, because it can store the images and instances on different physical servers.

- Telemetry service (originally ceilometer)

The Telemetry service provides user-level usage data, which can be used for customer billing, system monitoring, or alerts. It can collect data from notifications sent by OpenStack services such as Compute usage events, or by polling OpenStack infrastructure resources. Additionally, the service provides a plug-in system that can be used to add new monitoring metrics. The original Telemetry service has evolved to become three interconnected components.

- Alarming service (aodh): Enable the ability to trigger actions based on defined rules against sample or event data collected by Ceilometer.
- Timeseries database service (gnocchi): Storage and indexing of time series data and resources at a large scale. Useful in modern cloud platforms which are not only huge but also are dynamic and potentially multi-tenant. Gnocchi has been designed to handle large amounts of aggregates being stored while being performant, scalable and fault-tolerant. This component is no longer an OpenStack project, having grown to become an open-source project on its own.
- Event, metadata indexing service (panko): Panko is designed to provide a metadata indexing, event storage service which enables users to capture the state information of OpenStack resources at a given time. Its aim is to enable a scalable means of storing both short and long term data for use cases such as auditing and system debugging.
- Load Balancing service (octavia)

This service was designed to create a standalone load-balancing component, to replace the original networking server (neutron) LBaaS project which was based on HAProxy. The new service delivers load balancing services managed as virtual machines, containers, or bare metal servers (collectively known as amphorae) which are spun up on demand, providing stronger horizontal scaling.

- Orchestration service (heat)

The Orchestration service orchestrates multiple composite cloud applications using the Amazon Web Services (AWS) CloudFormation template format, through both a Representational State Transfer (REST) API and a CloudFormation-compatible Query API.

Additional Supported Services

Not all of the following additional services are covered in this course but are supported by Red Hat:

- Red Hat Ceph Storage (ceph)

Red Hat Ceph Storage is a distributed data object store designed to provide excellent performance, reliability, and scalability. It integrates with OpenStack services such as compute, block storage, shared file-system, image, identity, and object to give more flexibility storing images and volumes and operating at scale.

- Shared Filesystems service (manila)

The Shared File Storage service is a secure file share as a service. It uses the NFS and CIFS protocols for sharing file systems. It can be configured to run on a single-node back end or across multiple nodes.

- Big Data Processing Framework service (sahara)

The Big Data Processing Framework service aims to provide users with a simple means to provision a data processing cluster (such as Hadoop, Spark, and Storm) on OpenStack.

- Deployment service (tripleo)

The Deployment service code handles installing, upgrading, and operating OpenStack clouds using OpenStack's own services as the foundation. It uses Compute, Networking, and Orchestration services, and other orchestration tools, such as Puppet, and Ansible, to automate fleet management, including scaling out and scaling back at data center scale.

- Container Deployment service (kolla)

The service provides production-ready containers and deployment tools for operating OpenStack clouds that are scalable, reliable, and upgradable.

- Bare Metal Provisioning service (ironic)

The Bare Metal Provisioning service enables the user to provision physical hardware as opposed to virtual machines. It provides several drivers such as PXE and IPMI to cover a wide range of hardware. It also allows vendor-specific drivers to be added.

- Benchmarking service (rally)

The Benchmarking service is a benchmarking tool that automates and unifies multi-node Red Hat OpenStack Platform deployment, cloud verification, benchmarking, and profiling. It can be used as a basic tool for a system that would continuously improve its SLA, performance and stability.

- Integration Testing service (tempest)

The Integration Testing service provides a set of integration tests that run against a live OpenStack cloud. The Benchmarking service uses Integration Testing service as its verifier. It has several tests for OpenStack API validation, scenarios, and other specific tests useful in validating an OpenStack deployment.

- Messaging service (zaqar)

This is a multi-tenant cloud messaging service, for sending messages between various components of their applications. It is an efficient messaging engine designed with scalability and security in mind. Other OpenStack components integrate with Zaqr to surface events to end users and to communicate with guest agents that run in the overcloud.

OPENSTACK TERMINOLOGY

OpenStack uses the following terms:

Cloud controller

The coordinating manager. All machines in the OpenStack cloud communicate with the cloud controller using the REST API. Individual sub-components communicate with the Advanced Message Queuing Protocol (AMQP). In Red Hat OpenStack Platform, there are two options for AMQP: the Apache Qpid messaging daemon (**qpidd**), and RabbitMQ.

Cloud types

Cloud computing can be deployed in three forms: public, private, and hybrid clouds. These different forms, depending on the type of data being worked with, provides different levels of security and management.

A private cloud is a solution implemented by a single organization. It is mostly behind corporate firewalls and under internal IT control. This type of cloud is a good solution for speeding innovation, handling large compute and storage needs, or securing data.

Public cloud describes a cloud computing platform that is run and maintained outside of an organization infrastructure, based on shared systems. In this type of cloud, the public cloud service provider is in control of the entire infrastructure.

Hybrid cloud is a solution implemented to connect and use the resources provided by both private and public clouds. It allows management of the whole cloud infrastructure from a subset of tools under the control of corporate IT teams. The tools can access both types of clouds and distribute the compute load as needed. Hybrid clouds also provide tools for monitoring and self-servicing.

Cloud models

There are three models under which a cloud service can be categorized based on the service it delivers: Infrastructure-as-a-Service (IaaS), Platform-as-a-Service (PaaS), and Software-as-a-Service (SaaS).

- **Infrastructure-as-a-Service (IaaS)** is a service model that delivers compute infrastructure in a self-service model. Typically, IaaS provides compute hardware, storage, and networking components. Red Hat Ceph Storage (ceph)
- **Platform-as-a-Service (PaaS)** is a service model that delivers hardware and software tools usually required for application development in a self-service model. Using PaaS software developer are not required to deploy an development environment for developing or running a new software application.
- **Software-as-a-Service (SaaS)** is a software distribution service model which allows a software provider to host their applications and make them available over the internet.

Red Hat OpenStack Platform is based on Infrastructure-as-a-Service, which allows consumers to build their infrastructure for various deployments.

Red Hat OpenShift Container Platform is a Platform-as-a-Service, because it offers a scalable platform to host applications.

Compute node

A hypervisor; any machine running the compute service. The machine is running *only* the compute service, or it may have a *hyper-converged* configuration with collocated Ceph storage.

Host aggregates

A single Compute deployment can be partitioned into logical groups for performance or administrative purposes. A host aggregate creates logical units by grouping hosts. Host aggregates provide information used by the Compute scheduler (for example, limiting specific flavors or images to a subset of hosts). Administrators use host aggregates to handle load balancing, enforce physical isolation, and separate classes of hardware.

Availability zones

Availability zones are the end-user view of a host aggregate. The end user cannot see which hosts make up the zone, only the zone's name. End users can be directed to use specific zones that have been configured with certain capabilities or within certain areas. Administrators can use availability zones to spread their compute nodes geographically to achieve high availability.

Region

Regions can contain multiple availability zones. Each region within the cloud can have a separate set of Red Hat OpenStack Platform services. Different regions within the same cloud can share one Identity service server for authentication and one dashboard for administration.

Ephemeral disk or volume

A temporary disk used by an instance that is purposely discarded when the instances is terminated. Ephemeral disks may be located in local storage on a compute node, or stored in a back end such as Ceph. The purpose of ephemeral disks is to assist in self-cleaning server instances, which discard transient, runtime structures when the need for a server has ended. Ephemeral disks are used as root/boot disks, temporary workspace and for swap. Compare this to a persistent disk.

Persistent disk or volume

A disk design to remain and store data that survives the termination of server instance. Most applications and data require some form of persistent storage. When cloud instances are designed to emulate long-running legacy enterprise servers, their root disk is configured to be persistent instead of ephemeral. Persistent disks should never be used for transient data that is intended to be discarded with the server instance.

Instance

A virtual machine. Many utilities use the term *server*.

Stack

A group of instances built from a template. Template files are written in JavaScript Object Notation (JSON), a data exchange format designed to be a simpler alternative to Extensible Markup Language (XML) document encoding. Stacks and the template files are used in the orchestration service.

OpenStack networking (Neutron)

The OpenStack networking API uses the following abstractions to describe network resources:

- Network: An isolated L2 segment, analogous to a VLAN in the physical networking world.
- Subnet: A block of IPv4 or IPv6 IP addresses and associated configuration state.
- Port: A connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network. Also describes the associated network configuration, such as the MAC and IP addresses to be used on that port.

Open vSwitch

Software that provides a virtual switch. Open vSwitch provides traffic queuing and shaping and automated flow control. The Open vSwitch plug-in is used for OpenStack networking.



REFERENCES

Further information is available in the Components section of the *Product Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/product_guide/

DESCRIBING THE OPENSTACK ARCHITECTURE

Choose the correct answer to the following questions:

- ▶ 1. Which OpenStack service provides persistent volumes for instances?
 - a. Compute service
 - b. Image service
 - c. Dashboard
 - d. Block Storage service

- ▶ 2. Which OpenStack service provides images that are used as templates to build instances?
 - a. Compute service
 - b. Image service
 - c. Dashboard
 - d. Telemetry service

- ▶ 3. Which OpenStack service provides networking capabilities using a pluggable architecture?
 - a. Image service
 - b. Trove Database service
 - c. OpenStack Networking service
 - d. Compute service

- ▶ 4. Which OpenStack service provides a web dashboard for managing OpenStack?
 - a. Compute service
 - b. Orchestration service
 - c. Dashboard
 - d. Block Storage service

- ▶ 5. Which OpenStack service provides authentication and authorization?
 - a. Compute service
 - b. Object Store service
 - c. Identity service
 - d. Bare Metal service

► **6. Which OpenStack service provides virtualization using libvirt, qemu, and kvm?**

- a. Compute service
- b. Object Store service
- c. Identity service
- d. Bare Metal service

DESCRIBING THE OPENSTACK ARCHITECTURE

Choose the correct answer to the following questions:

- ▶ 1. Which OpenStack service provides persistent volumes for instances?
 - a. Compute service
 - b. Image service
 - c. Dashboard
 - d. Block Storage service

- ▶ 2. Which OpenStack service provides images that are used as templates to build instances?
 - a. Compute service
 - b. Image service
 - c. Dashboard
 - d. Telemetry service

- ▶ 3. Which OpenStack service provides networking capabilities using a pluggable architecture?
 - a. Image service
 - b. Trove Database service
 - c. OpenStack Networking service
 - d. Compute service

- ▶ 4. Which OpenStack service provides a web dashboard for managing OpenStack?
 - a. Compute service
 - b. Orchestration service
 - c. Dashboard
 - d. Block Storage service

- ▶ 5. Which OpenStack service provides authentication and authorization?
 - a. Compute service
 - b. Object Store service
 - c. Identity service
 - d. Bare Metal service

► **6. Which OpenStack service provides virtualization using libvirt, qemu, and kvm?**

- a. Compute service
- b. Object Store service
- c. Identity service
- d. Bare Metal service

INTRODUCING LAUNCHING AN INSTANCE

PERFORMANCE CHECKLIST

In this lab, you will delete the instance created by the setup script, launch an instance, verify that the created instance uses the correct image and flavor, and verify that the instance is up and running. You can perform the tasks in this lab using either the dashboard or the OpenStack unified CLI.

OUTCOMES

You should be able to:

- Delete an instance.
- Create a new instance.
- Verify the newly created instance.

Ensure that **workstation**, **director**, and the overcloud's virtual machines are started. If not already logged in, log in to **workstation** as **student** using **student** as the password.

From **workstation**, run **lab launch-lab setup** script with the **setup** argument, which creates the required components for this exercise, such as users, project, images, and networks for launching an instance.

```
[student@workstation ~]$ lab launch-lab setup
```

1. To perform the lab tasks using the dashboard, open Firefox on **workstation** and browse to <http://dashboard.overcloud.example.com>.
To perform the lab tasks using the OpenStack unified CLI, on **workstation**, open a terminal.
2. Log in to the dashboard using **operator1** as the user name and **redhat** as the password.
Alternatively, to use the OpenStack unified CLI, source the **operator1** credential file / **home/student/operator1-production-rc** from the command line.
3. Delete the existing instance named **production-server2**.
4. Launch an instance named **production-server1** using the **default** flavor, the **production-network1** network and the **rhel7** image.
5. Connect to the console of **production-server1**.
6. Verify the instance from the console of **production-server1** and use the **ping** command to reach the DHCP server.

Evaluation

On **workstation**, run the **lab launch-lab grade** command to confirm the success of this exercise.

```
[student@workstation ~]$ lab launch-lab grade
```

Cleanup

On **workstation**, run the **lab launch-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab launch-lab cleanup
```

This concludes the lab.

INTRODUCING LAUNCHING AN INSTANCE

PERFORMANCE CHECKLIST

In this lab, you will delete the instance created by the setup script, launch an instance, verify that the created instance uses the correct image and flavor, and verify that the instance is up and running. You can perform the tasks in this lab using either the dashboard or the OpenStack unified CLI.

OUTCOMES

You should be able to:

- Delete an instance.
- Create a new instance.
- Verify the newly created instance.

Ensure that **workstation**, **director**, and the overcloud's virtual machines are started. If not already logged in, log in to **workstation** as **student** using **student** as the password.

From **workstation**, run **lab launch-lab setup** script with the **setup** argument, which creates the required components for this exercise, such as users, project, images, and networks for launching an instance.

```
[student@workstation ~]$ lab launch-lab setup
```

1. To perform the lab tasks using the dashboard, open Firefox on **workstation** and browse to <http://dashboard.overcloud.example.com>.
To perform the lab tasks using the OpenStack unified CLI, on **workstation**, open a terminal.
2. Log in to the dashboard using **operator1** as the user name and **redhat** as the password.
Alternatively, to use the OpenStack unified CLI, source the **operator1** credential file `/home/student/operator1-production-rc` from the command line.
 - 2.1. Enter **operator1** for the user name and **redhat** as the password.
To perform the task using the command line, source the **operator1** credential file. The credential file sets Identity service authentication endpoint, user name, password, domain, and project to be used by the OpenStack unified CLI.

```
[student@workstation ~]$ source ~/operator1-production-rc  
[student@workstation ~(operator1-production)]$
```

3. Delete the existing instance named **production-server2**.
 - 3.1. To perform this task using the dashboard, navigate to the Instances subtab under the Compute tab.
 - 3.2. On the Instances subtab, in the **production-server2** row, click the Actions menu. Click Delete Instance.
 - 3.3. Click Delete Instance to confirm.

- 3.4. Alternatively, use the **openstack server list** command to list all available instances.

```
[student@workstation ~(operator1-production)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "production-server2",
    "Image": "rhel7",
    "ID": "302be439-5032-4a8d-a306-2cf7d1a6faff",
    "Flavor": "default",
    "Networks": "production-network1=192.168.1.1"
  }
]
```

- 3.5. Use the **openstack server delete** command with the name **production-server2** or the ID from the previous step to delete the instance.

```
[student@workstation ~(operator1-production)]$ openstack server delete \
production-server2
```

4. Launch an instance named **production-server1** using the **default** flavor, the **production-network1** network and the **rhel7** image.
- 4.1. To perform this task using the dashboard, navigate to the Instances subtab under the Compute tab.
 - 4.2. Click Launch Instance.
 - 4.3. In the Details tab, enter **production-server1** as the Instance Name.
 - 4.4. In the Source tab, select Instance Boot Source → Image.
 - 4.5. To allocate the rhel7 image, on the Source tab, click the up arrow button on the same row as the rhel7 image. Click the No button under Create New Volume.
 - 4.6. To allocate the default flavor, on the Flavor tab, click the up arrow button on the same row as the **default** flavor.
 - 4.7. Click Launch Instance to launch the **production-server1** instance.
 - 4.8. Alternatively, use the **openstack server create** command to create the **production-server1** instance.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--image rhel7 \
--flavor default \
--wait production-server1
+-----+
| Field          | Value        |
+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2018-06-04T13:10:17.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4      |           |
| accessIPv6      |           |
```

addresses	production-network1=192.168.1.X
adminPass	khdcFj44rQA4
config_drive	
created	2018-06-04T13:09:50Z
flavor	default (677cc5e4-fd40-4689-a49d-e8382403d609)
hostId	5406...fce0
id	18f32508-7348-48d2-903a-58b95e8eea75
image	rhel7 (30af028a-0aa0-424c-b57c-436c15c89c0e)
key_name	None
name	production-server1
progress	0
project_id	d1d0a0f55d5e47c7a46f1a7e689a7544
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-04T13:10:17Z
user_id	620ad3829dc4f31af7abde5d7571d6c
volumes_attached	

5. Connect to the console of **production-server1**.

- 5.1. Navigate to the Instances subtab under the Compute tab.
- 5.2. Right-click the **production-server1** instance link and choose Open Link in New Tab. If a certificate error appears, accept the self-signed certificate.
- 5.3. In the new tab, choose the Console tab, then click the Click here to show only console link. Watch the virtual machine boot (it may have already booted).
- 5.4. Alternatively, use the **openstack console url show** command to view the console URL from the CLI.

```
[student@workstation ~-(operator1-production)]$ openstack console url show \
production-server1
+-----+
| Field | Value
+-----+
| type  | novnc
| url   | https://172.25.250.50:6080/vnc_auto.html?token=72f431(...)da10a
+-----+
```

5.5. Open the URL using Firefox to access the VNC instance console.

6. Verify the instance from the console of **production-server1** and use the **ping** command to reach the DHCP server.
- 6.1. To verify the setup, log in to the **production-server1** instance as the **root** user with **redhat** as the password.

```
Red Hat Enterprise Linux Server 7.5 (Maipo)
Kernel 3.10.0-862.el7.x86_64 on an x86_64

small image
production-server1 login: root
Password: redhat
```

```
[root@production-server1 ~]#
```

6.2. Ping the DHCP server from **production-server1**.

```
[root@production-server1 ~]# ping -c3 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.642 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.457 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.596 ms
--- 192.168.1.2 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1000ms
rtt min/avg/max/mdev = 0.457/0.565/0.642/0.078 ms
```

Evaluation

On **workstation**, run the **lab launch-lab grade** command to confirm the success of this exercise.

```
[student@workstation ~]$ lab launch-lab grade
```

Cleanup

On **workstation**, run the **lab launch-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab launch-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Red Hat OpenStack Platform is comprised of services including dashboard, identity service, compute service, block storage service, object store, OpenStack networking service, image service, orchestration service, load balancing service, and telemetry Service.
- The dashboard can be used to launch instances and to create resources in Red Hat OpenStack Platform.
- The unified CLI can be used to launch instances and to create resources in Red Hat OpenStack Platform.

CHAPTER 2

ORGANIZING PEOPLE AND RESOURCES

GOAL

Manage projects, users, roles, and quotas.

OBJECTIVES

- Manage OpenStack projects.
- Manage OpenStack user accounts.
- Assign OpenStack user roles and privileges.
- Manage quotas for projects.

SECTIONS

- Managing Projects (and Guided Exercise)
- Administering Users (and Guided Exercise)
- Assigning User Roles and Privileges (and Guided Exercise)
- Managing Quotas (and Guided Exercise)

LAB

Organizing People and Resources

MANAGING PROJECTS

OBJECTIVE

After completing this section, students should be able to manage OpenStack projects.

INTRODUCTION TO THE IDENTITY SERVICE

In Linux, users and groups are implemented to control access to operating system resources, like directories, files, and peripherals. A user is anyone who uses the computer, and managing these users is part of securing and limiting access to these resources. Red Hat OpenStack Platform uses domains, projects, groups, and users to organize OpenStack resources. The **admin** user can be used to log in to the Dashboard or to manage OpenStack using the command line. This **admin** user is similar to the **root** user in the Linux operating system. This user has a role in the **admin** project.

In OpenStack, the authentication and authorization services are provided by the OpenStack identity service, code-named *Keystone*. Users authenticate via the identity service API endpoint before they request other OpenStack services. Red Hat OpenStack Platform supports both version 2 and 3 of the identity API. APIv3 of the identity service includes new features such as domains and groups. To use the APIv3 endpoint, users must set the `OS_IDENTITY_API_VERSION` variable, and modify the `OS_AUTH_URL` variable in the identity environment file.

```
export OS_USERNAME=user-name
export OS_PASSWORD=password
export OS_PROJECT_NAME=project-name
export OS_IDENTITY_API_VERSION=3
export OS_AUTH_URL=http://AUTH-ENDPOINT:5000/v3
```

TERMS USED WITH IDENTITY SERVICE

Projects

A project is a collection of resources, such as networks, images and instances. Depending on the OpenStack implementation, projects can map to a customer, an account, an organizational unit, or a development project. Red Hat OpenStack Platform installs with two default projects named **admin** and **service**. These projects exist in the default domain named **default**.

Users

Users represent the end user, operator or administrator who uses the OpenStack services. Red Hat OpenStack Platform installs with a default **admin** user with an **admin** role assigned in the **admin** project.

Roles

Roles define a set of user privileges to perform specific actions on OpenStack services. Red Hat OpenStack Platform installs with several default roles, including **admin** and **_member_**. The **_member_** role provides normal user access to all of a project's resources. The **admin** role provides additional administrative privileges throughout the user's domain.

Authentication parameters

Authentication parameters are the parameters required by the identity service to authenticate a user. This information includes a user name, password, project name, region name, and identity service URL endpoint. Using this information, when a user requests to access a

service, the identity service provides an authentication token that enables the user to access the service.

Regions

Regions are used to logically represent different Red Hat OpenStack Platform deployments. These deployments can be separated by geographic location and can be named accordingly. For example, a region can be named **in-mumbai** to represent the OpenStack deployment at Mumbai, India. These regions can further be subdivided into subregions.

Domains

Domains are collections of projects and users that can be defined as a single identity realm. Resources are not shared or moved between domains. Domains are supported with version 3 of the identity service API. If no domains are created, all projects and users use the **default** domain.

Groups

Groups are collections of users within a domain. Privileges can be assigned to all the users in a group by assigning a role to a group. These privileges and the association of a user to a project or domain can be revoked by removing the user from the group. Groups are supported in version 3 of the identity service API.

MANAGING OPENSTACK PROJECTS

A project is a collection of resources, like networks, images or instances. Projects help users to use a single OpenStack environment without interfering with other project resources. For example, an image uploaded to a project can be marked as private, so that project users cannot access the image. In earlier versions of Red Hat OpenStack Platform, the word tenant was used interchangeably with project.

One or many users can be assigned to a project. A user assigned to a project can create and manage virtual resources within the project. Users can be assigned to one or many projects simultaneously.

Creating and Deleting Projects

Projects can be created by any user assigned the **admin** role. Projects are configured with resource quotas that can be edited when creating the project or modified later. All user associations to a project, and all project resources, are removed when the project is deleted.

Creating and Deleting Projects Using the Dashboard

The following steps describe the process of creating and deleting a project using the dashboard.

1. To create a project, open the dashboard in a web browser, and log in as a user with administrator privileges. Navigate to **Identity** → **Projects** and click **Create Project**.
2. In the **Project Information** tab, populate the **Name** field with the project name. Optionally, populate the **Description** field with a short description of the project. Select the **Enabled** check box to enable the project. To disable a project, clear the **Enabled** check box.
3. Click **Create Project** to create the project.
4. To delete a project with the dashboard, navigate to **Identity** → **Projects**, and select the project to be deleted. Click **Delete Projects**, and then click **Delete Projects** again to confirm the deletion.

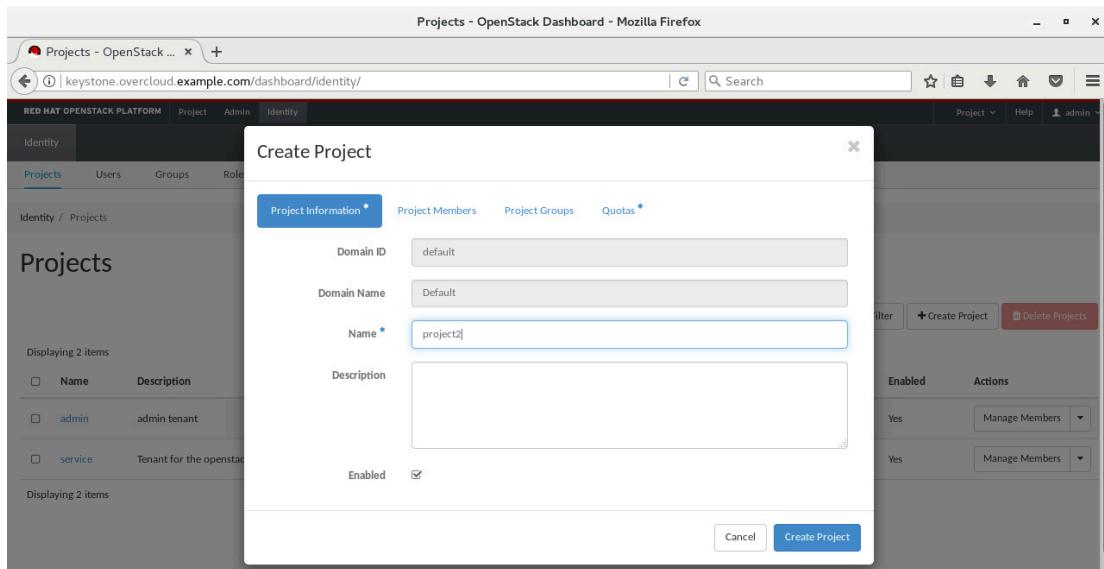


Figure 2.1: Creating projects in the Dashboard

The OpenStack unified CLI can be used to create and delete projects from the command line. The **openstack project** command provides project management features; projects can be tagged, disabled, enabled, or the project name and description edited using the **openstack project set** command. To create a project named **demo-project** in the **demo-org** domain, use the following command:

```
[user@demo ~]# openstack project create \
--description "Demo Org Project" --domain demo-org demo-project
```

Creating and Deleting Projects Using the OpenStack CLI

The following steps outline the process for creating and deleting a project using the OpenStack unified CLI.

1. Source the identity environment file so that the OpenStack unified command line tool can communicate with the OpenStack services. To create a project, the user must have administrator privileges.
2. Create the project using the **openstack project create** command. The command takes the project name as an argument. You can use the **--description** option to specify the description for the project, but this is optional. The project is enabled by default, but the **--disable** option can be used to start with the project disabled.
3. Delete the project using the **openstack project delete** command. The command takes the project name or ID as an option. You can delete multiple projects at the same time with a space separated list.
4. Use the **openstack project list** command to list all currently available projects, and to verify that the project was deleted.

Projects and Their Use Cases

Projects support different use cases depending upon how users are assigned:

- Many users in a project: This use case is commonly used when users in a project can create several cloud resources and isolate those resources within the project.

- One user in one project: This use case is required when multiple cloud users work on the same OpenStack environment, but have distinct roles defined. For example, the **network-operator** user is responsible for creating the networks and providing access to other projects. The **cloud-operator** user could have a role assigned in another project and can only launch an instance using the network created by the **network-operator** user.
- One user assigned to several projects: This use case is used when there is a requirement for a user who can switch roles to perform several tasks across multiple projects. For example, a user as a team leader for a project wants access to his private project and the projects he leads. When a user is assigned multiple projects, one of the projects can be marked as the primary project.

Default Projects and Users

Red Hat OpenStack Platform ships with two default projects: **admin** and **service**. The **admin** project has a single user assigned by default; the **admin** user account. The **service** project has multiple users by default; one for each OpenStack component service installed. These service user accounts are intended for secure component communications and are not intended to be used as normal user accounts. Because the **admin** user is assigned the **admin** role, this user has full access to all projects within the OpenStack environment. The **admin** user is commonly used to create new projects, add users to those projects, assign roles to users, and to create certain shared networks, images and flavors in the **admin** project. The **admin** user should not use the **admin** project to launch instances.

Domains

Domains represent collections of projects, users, and groups owned by a specific identity domain. These identity domains can be mapped to organizations or customers. Users can be associated with multiple projects associated to multiple domains.

For example, a cloud provider can host various customers, and each customer requires different authentication back ends. These customers' cloud resources can be segregated using domains. Domains help the customer to choose between the various authentication back ends supported by the cloud provider. Domains also help the cloud provider to host all the customers using a single OpenStack environment. By default, the dashboard has multiple domains support disabled.

Various authentication back ends such as Microsoft Active Directory Domain Service (AD DS), Red Hat Identity Management (IdM), and LDAP can be used to integrate with the OpenStack identity service. Separate domains must be configured to authenticate users in different LDAP or AD DS environments. Red Hat OpenStack Platform uses these authentication back ends for user account authentication (identity), while it retains authorization management (access) for the authenticated users.



REFERENCES

Additional information may be available in the section on Project Management in the *Users and Identity Management Guide* for Red Hat OpenStack Platform, which can be found at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/users_and_identity_management_guide/

MANAGING PROJECTS

In this exercise, you will create and delete projects using the dashboard and the OpenStack unified CLI.

OUTCOMES

You should be able to:

- Create and delete projects using the dashboard.
- Create and delete projects using the OpenStack unified CLI.

Ensure that the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using the password **student**. From **workstation**, run the **lab manage-projects setup** command. The lab script verifies the OpenStack environment, and checks that the environment file for the **admin** user is available.

```
[student@workstation ~]$ lab manage-projects setup
```

- ▶ 1. On **workstation**, open Firefox and browse to `http://dashboard.overcloud.example.com`.
- ▶ 2. Log in to the dashboard using **admin** as the user name and **redhat** as the password.
- ▶ 3. Use the dashboard to create the **research** project using default project quotas.
 - 3.1. Navigate to Identity → Projects and click **+Create Project**.
 - 3.2. Enter **research** in the Name field and **Research Project** in the Description field. Leave the Enabled check box selected.Click **Create Project** and then log out of the dashboard.
- ▶ 4. On **workstation** as the user **student**, use the OpenStack unified CLI to create the **sales** project from the command line. Use the `/home/student/admin-rc` environment file to access OpenStack as the **admin** user.
 - 4.1. On **workstation**, from the command line, source the identity environment file for the **admin** user to gain administrator rights. The environment file sets the OpenStack identity authentication endpoint, user name, password, region, and project for the OpenStack unified CLI.

```
[student@workstation ~]$ source ~/admin-rc  
[student@workstation ~]$
```

- 4.2. Create the **sales** project. Use **Sales Project** as the description.

```
[student@workstation ~]$ openstack project create \
```

```
--description "Sales Project" \
sales
+-----+
| Field      | Value
+-----+
| description | Sales Project
| domain_id   | default
| enabled     | True
| id          | a141c847e10448568f71d5091613b506
| is_domain   | False
| name        | sales
| parent_id   | default
| tags        | []
+-----+
```

- ▶ 5. Verify that the **sales** project has been created.

```
[student@workstation ~]$ openstack project list
+-----+
| ID           | Name
+-----+
| 76c7203da79a423e9676711f4c880aa3 | sales
| 6e47d5fd39c6444b959f08d2afbe4342 | research
... output omitted ...
+-----+
```

- ▶ 6. Delete the **sales** project.

- 6.1. Delete the **sales** project using the command line.

```
[student@workstation ~]$ openstack project delete sales
```

- 6.2. Verify that the **sales** project has been deleted.

```
[student@workstation ~]$ openstack project list
+-----+
| ID           | Name
+-----+
| 76c7203da79a423e9676711f4c880aa3 | research
... output omitted ...
+-----+
```

Cleanup

From **workstation**, run the **lab manage-projects cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab manage-projects cleanup
```

This concludes the guided exercise.

ADMINISTERING USERS

OBJECTIVE

After completing this section, students should be able to manage OpenStack user accounts.

INTRODUCTION TO OPENSTACK USERS

In all software, system users can be either real people or accounts configured for automated application access. OpenStack end users are those that use the self-service interface to create and consume instances, volumes, networks, and images, and perform other tasks at the project user level. The service accounts in the **service** project are also normal user accounts, but are only used by the services themselves for some specific inter-service authentication. Users can access the resources belonging to the project in which they are assigned. OpenStack users have a limited set of privileges defined by the role(s) assigned to the user.

Authentication is the process of confirming the user identity. Typically, a user name and password is initially needed to log in to the authentication interface. The Red Hat OpenStack Platform identity service supports multiple methods of authentication, including user name and password, LDAP, and other external authentication methods. Upon successful authentication, users are provided with an authorization token to access OpenStack services.

OpenStack Groups

Groups were introduced in the identity service APIv3, as a collection of users in a domain. Similar to Linux groups, a role assigned to a group applies to all users in that group. The identity service APIv3 allows you to create groups using the dashboard or the OpenStack unified CLI. Use the **openstack group create** command to create groups. The **--domain** option is used to create a group in a specific domain. To create a group named **demo-group** in the **demo-org** domain, use the following command:

```
[user@demo ~]$(admin-admin)]$ openstack group create --domain demo-org demo-group
```

MANAGING OPENSTACK USERS

Red Hat OpenStack Platform provides a default user named **admin** with the **admin** role who has full privileges to access all projects and domains. Using the **admin** privileges, a user can create or edit other users, and add or remove them from projects or groups in the current domain, using either the dashboard or OpenStack unified CLI. The dashboard has multiple domain support disabled by default and uses a single domain called **default**. A user can have a role in multiple projects and can be members of multiple groups in a single domain, but must have a unique user account for each domain for which they will be given access.

Managing Users from the Dashboard

Creating a user using the dashboard requires administrator privileges. The dashboard user creation form provides fields to select a user's primary project, password, and email address.

An existing user can be disabled to block account use temporarily, by any user with an **admin** role. On the dashboard, navigate to Identity → Users. Disable the user by selecting Disable User from the list under the Actions column.

On the dashboard, on the Users tab, enable the user by selecting Enable User from the list under the Actions column.

After creating the user, enter the user name and password on the dashboard login screen to access the OpenStack services.

Managing Users from the Dashboard

The following steps outline the process for creating, deleting, and editing users from the dashboard.

1. To create a user, open the dashboard in a web browser, and log in as a user with administrator privileges. Navigate to Identity → Users and click Create User.
2. Populate the Name field with the user name, and the Password and Confirm Password fields with the password. Select the project name from the Primary Project list to add the user. The Role list defines the role assigned to the user. Optionally, populate the Email and Description fields. To temporarily disable the user, clear the Enabled check box.
3. Click Create User.
4. Verify the newly created user by logging in to the dashboard with the new user name and password.
5. To disable a user account, log in to the dashboard as an administrator. Navigate to the Identity → Users subtab.

Disable the user by selecting Disable User from the list under the Actions column.

The disabled user will not be able to log in to the dashboard.

6. To enable a user account, navigate to the Users tab for the project. Enable the user by selecting Enable User from the list under the Actions column.
7. To change the password of a user, select Change Password from the list under the Actions column.

In the dialog box, enter the new password in the Password and Confirm Password fields. Click Save.

8. To delete a user, navigate to Identity → Users and select Delete User from the list under the Actions column for that user. Click Delete User again to confirm.

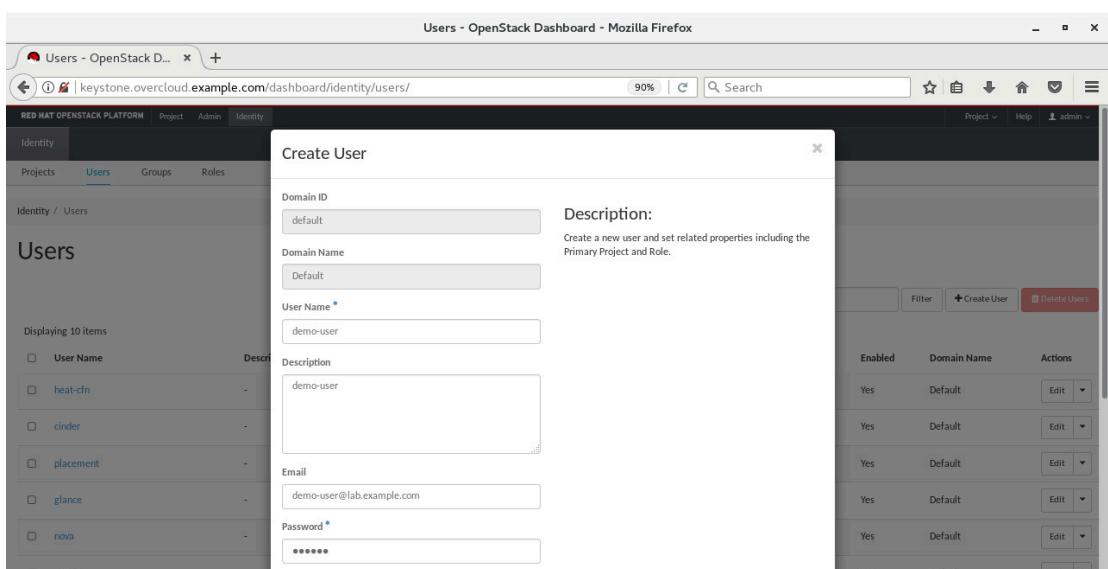


Figure 2.2: Creating a user in the Dashboard

Managing Users from the CLI

To create a user from the OpenStack CLI, use the **openstack user create** command. To create a user whose password needs to be entered interactively, use the **--password-prompt** option. When using the OpenStack command-line clients, the identity environment file for the user must be sourced. The user environment file sets the required environment variables for the OpenStack command-line clients.

```
[user@demo ~(admin-admin)]$ openstack user create --project demoproject \
--password-prompt demouser
User Password: password
Repeat User Password: password
+-----+
| Field          | Value           |
+-----+
| default_project_id | 52dfb66385344f14b7ed3e837163a484 |
| domain_id      | default          |
| enabled         | True             |
| id              | 3a1ab09c9d384f3a99b9aa4bf4dd59b9 |
| name            | demouser         |
| options          | {}               |
| password_expires_at | None            |
+-----+
```

The **--or-show** option, when used with the **openstack user create** command, displays the user's details if the user already exists. This option is useful for idempotent functions where the **openstack user create** command is run multiple times.

```
[user@demo ~(admin-admin)]$ openstack user create --project demo-project \
--password-prompt --or-show demouser
User Password: password
Repeat User Password: password
+-----+
| Field          | Value           |
+-----+
| default_project_id | 52dfb66385344f14b7ed3e837163a484 |
| domain_id      | default          |
| enabled         | True             |
| id              | 3a1ab09c9d384f3a99b9aa4bf4dd59b9 |
| name            | demouser         |
| options          | {}               |
| password_expires_at | None            |
+-----+
```

To add a user to a group when using identity service APIv3, use the **openstack group add user** command. Use the **openstack group contains user** to verify that a user is a member of a group.

With the OpenStack CLI, use the **openstack user set** command as an administrator. The **--disable** option is added to disable the user account.

```
[user@demo ~(admin-admin)]$ openstack user set --disable demouser
```

To enable the user from the command line, use the **openstack user set --enable** command.

```
[user@demo ~(admin-admin)]$ openstack user set --enable demouser
```

Managing Users from the OpenStack CLI

The following steps outline the process for creating, deleting, and editing a user from the OpenStack CLI.

1. Source the project administrator environment file to gain administrator privileges.
2. Use the **openstack user create** command to create a user in the existing project.
3. Use the **openstack role add** command to assign the **_member_** user role to the new user in the existing project.
4. Use the **openstack user list** command to list all users. Use the **openstack user show** command to see the details of a specific user.
5. Verify the new user can access OpenStack. Create an environment file for the user, and source it. Then, run any **openstack** command using the authentication parameters of the user.
6. As an administrator, use the **openstack user set --disable** command to disable a user account.
7. Verify that the user account is disabled. Source the environment file of the disabled user, and then execute any **openstack** command.
8. As an administrator, update the user's account. Use the **openstack user set** command to enable the user or update their email address, user name, or password. The **--enable** option is used to enable the user account.

The **--password-prompt** option prompts for the password interactively.

9. Use the **openstack user delete** command to delete the user. The user name or ID must be specified as an argument to the command.
10. Confirm the user is deleted by listing the current users in the project.

Requesting an Authorization Token using the authentication parameters

Using the **openstack token issue** command a user is issued an authorization token after validating the user's authentication parameters. The authorization token is valid for a limited period as defined in the **expires** field. With the authorized token, the user can request service tasks, such as creating and managing instances, volumes, networks, images, and other tasks. An application using the service REST APIs, can use an authorization token to perform service tasks. Compromised tokens can be revoked. To revoke a token before it expires, use the **openstack token revoke** command.

```
[user@demo ~(admin-admin)]$ openstack token issue
+-----+-----+
| Field      | Value
+-----+-----+
| expires① | 2018-05-31T12:10:15+0000
| id②       | 90a3349d3e81413e2be1f3745b54
| project_id | 5745b52a6c45440ea693ce075c8ee757
| user_id    | f95d249a757f47879503c51a305f9534
```

- +-----+-----+
- ① Token expiry date
 - ② ID of the token issued to the user

Use the generated token to retrieve information about the project on which the authorized user has a role assigned. The following example shows the **curl** command using the OpenStack identity API and the token ID to list all projects to which the user has a role assigned. The final pipe uses **python** to format the JSON output.

```
[user@demo ~]$(admin-admin)]$ curl -s -H "X-Auth-Token: 90a3349d3e81413e2be1f3745b54" \
http://172.25.250.50:5000/v3/projects | python -m json.tool
{
    ...output omitted...
    "projects": [
        {
            "description": "Demo Project",
            "domain_id": "897a31bedfcc449eb10f6e2c3568513",
            "enabled": true,
            "id": "52dfb66385344f14b7ed3e837163a484",
            "is_domain": false,
            "links": {
                "self": "http://172.25.250.50:5000/v3/projects/52df...a484"
            },
            "name": "demoproject",
            "parent_id": "897a31bedfcc449eb10f6e2c3568513",
            "tags": []
        }
    ...
    ],
}
```

Adding a User to a Project

User and groups can be added to a project while it is being created. Existing users can also be added to a project using the dashboard or the OpenStack CLI. From the dashboard, click Identity → Projects → Manage Members to add or remove users from a project.

The screenshot shows the Red Hat OpenStack Platform Identity dashboard. On the left, there's a sidebar with 'Projects' and 'Identity' sections. The main area has tabs for 'Project Information', 'Project Members' (which is selected), 'Project Groups', and 'Quotas'. In the 'Project Members' section, there's a table for 'All Users' and another for 'Project Members'. The 'Project Members' table currently has one row for 'demouser'. At the bottom right of the main content area, there are 'Cancel' and 'Save' buttons.

Figure 2.3: Adding users to a project from the dashboard

In the OpenStack CLI, use the **openstack user set --project** command to change the primary project of a user. To add a user to an additional project, use the **openstack role add** command. Use this command to assign the `_member_` role for the user in a project. The following shows a user named **demouser** being assigned the `_member_` role in the **demoproject** project.

```
[user@demo ~]# openstack role add --project demoproject --user demouser _member_
```

Roles are explained in more detail in the following section.



REFERENCES

Additional information may be available in the section on User Management and Group Management in the *Users and Identity Management Guide* for Red Hat OpenStack Platform, which can be found at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/users_and_identity_management_guide/

ADMINISTERING USERS

In this exercise, you will use the dashboard and the OpenStack unified CLI to administer user accounts.

OUTCOMES

You should be able to:

- Create users.
- Edit a user account.
- Add users to a project.
- Delete a user account.
- Enable and disable a user account.
- Verify the created users.

Ensure that the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using the password **student**. On the **workstation** machine, execute the **lab administer-users** script using the **setup** argument. The lab script ensures that the **finance** project and the environment file for the **admin** user exists.

```
[student@workstation ~]$ lab administer-users setup
```

- ▶ 1. On **workstation**, open Firefox and browse to <http://dashboard.overcloud.example.com>.
- ▶ 2. Log in to the dashboard using the **admin** user and **redhat** as the password.
- ▶ 3. Create the **developer2**, **developer3**, and **developer4** users using the dashboard. Set the password for these user accounts to **redhat**. Add the users to the **finance** project.
 - 3.1. Navigate to Identity → Users and click +Create User.
 - 3.2. In the dialog box, enter **developer2** for the User Name field, and **redhat** in the Password and Confirm Password fields. Select the **finance** project from the Primary Project list. Leave the Role set to **_member_**, and the Enabled check box selected. Click Create User.
 - 3.3. Repeat the steps for the **developer3** and **developer4** users.
- ▶ 4. Delete the **developer3** user account using the dashboard.
 - 4.1. On the Users tab, select the **developer3** user account. Click Delete Users and then click Delete Users again to confirm.

- ▶ 5. Disable the **developer4** user account and verify that the user can not log in to the dashboard.
 - 5.1. On the Users tab, select Disable User for the **developer4** user, under the Actions column.
 - 5.2. Verify that the **developer4** user can not log in to the dashboard.
Log out of the dashboard as **admin**, and then log in as **developer4** using the password **redhat**. You should not be able to log in as the **developer4** user.
- ▶ 6. Enable the **developer4** user account using the dashboard.
 - 6.1. Log in to the dashboard as the **admin** user.
 - 6.2. Navigate to Identity → Users, and select Enable User for the **developer4** user, under the Actions column.
- ▶ 7. Change the password for the **developer4** user to **password**.
 - 7.1. On the Users tab, select Change Password for the **developer4** user, under the Actions column.
In the dialog box, enter **password** in the Password and Confirm Password fields, and then click Save.
- ▶ 8. Verify that the **developer4** user can log in to the dashboard with the password **password**.
 - 8.1. Log out as the **admin** user.
Log in to the dashboard as the **developer4** user using the password **password**. The **developer4** user will be able to successfully log in to the dashboard if the user account is enabled.
 - 8.2. Log out of the dashboard as the **developer4** user.
- ▶ 9. Create the **developer5** user as a member of the **finance** project using the OpenStack CLI. Set the password to **redhat** and the email address to **user@workstation.lab.example.com**.
 - 9.1. Log in to **workstation** as student. Source the identity environment file to provide administrator rights to the **admin** user. The **/home/student/admin-rc** file sets the OpenStack identity service authentication endpoint, user name, password, region, and project used by the OpenStack CLI.

```
[student@workstation ~]$ source ~/admin-rc  
[student@workstation ~]$
```

- 9.2. Create the **developer5** user as a member of the **finance** project. Assign the **_member_** role to the **developer5** user in the **finance** project. Set the password to **redhat** and the email address to **user@workstation.lab.example.com**.

```
[student@workstation ~]$ openstack user create \  
--project finance \  
--password redhat \  
--email user@workstation.lab.example.com \  
developer5  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| default_project_id | 4848228431d44e3a9373fa0547f50bdb |  
| domain_id | default |
```

```
| email           | user@workstation.lab.example.com   |
| enabled         | True                           |
| id              | 77c286549bf94507abebdc9ac5e00b6f |
| name            | developer5                     |
| options          | {}                            |
| password_expires_at | None                         |
+-----+-----+
```

9.3. Assign the `_member_` role to the `developer5` user in the `finance` project.

```
[student@workstation ~]$ openstack role add \
--project finance \
--user developer5 \
_member_
```

9.4. Verify that the `developer5` user has been created.

```
[student@workstation ~]$ openstack user list
+-----+-----+
| ID           | Name        |
+-----+-----+
... output omitted ...
| 77c286549bf94507abebdc9ac5e00b6f | developer5 |
... output omitted ...
+-----+-----+
```

▶ 10. Create the `/home/student/developer5-finance-rc` environment file for the `developer5` user.

10.1. Copy the existing `/home/student/admin-rc` environment file to `/home/student/developer5-finance-rc`.

```
[student@workstation ~]$ cp ~/admin-rc ~/developer5-finance-rc
```

10.2. Edit the `/home/student/developer5-finance-rc` file, and modify the `OS_USERNAME`, `OS_PASSWORD`, `OS_PROJECT_NAME`, and `PS1` variables to read as follows:

```
export OS_USERNAME=developer5
export OS_PASSWORD=redhat
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export PS1='[\u@\h \w(developer5-finance)]\$ '
export OS_PROJECT_NAME=finance
...output omitted...
```

▶ 11. Disable the `developer5` user account using the OpenStack CLI.

11.1. Disable the `developer5` user account.

```
[student@workstation ~]$ openstack user set \
--disable developer5
```

▶ 12. Verify that the **developer5** user account is disabled.

- 12.1. Source the **/home/student/developer5-finance-rc** environment file for the **developer5** user.

```
[student@workstation ~]$ source ~/developer5-finance-rc  
[student@workstation ~(developer5-finance)]$
```

- 12.2. Run the **openstack flavor list** command. Take note of the message returned in the response.

```
[student@workstation ~(developer5-finance)]$ openstack flavor list  
The request you have made requires authentication. (HTTP 401) (Request-ID:  
req-ab5de0a1-106b-4d20-898b-3a785e3c956a)
```

▶ 13. Enable the **developer5** user and change the password to **password**. Edit the email address of the **developer5** user to **student@workstation.lab.example.com** using the OpenStack CLI.

- 13.1. Source the identity environment file to provide administrator rights to the **admin** user.

```
[student@workstation ~(developer5-finance)]$ source ~/admin-rc
```

- 13.2. Enable the **developer5** user, change the password to **password**, and modify the email address to **student@workstation.lab.example.com**.

```
[student@workstation ~(developer5-finance)]$ openstack user set \  
--password password \  
--email student@workstation.lab.example.com \  
--enable developer5
```

▶ 14. Verify that the **developer5** user account is enabled and uses the password set for authentication.

- 14.1. In the **/home/student/developer5-finance-rc** environment file, modify the **OS_PASSWORD** variable value to **password**.

```
export OS_USERNAME=developer5  
export OS_PASSWORD=password  
export OS_AUTH_URL=http://172.25.250.50:5000/v3  
export PS1='[\u@\h \w(developer5-finance)]\$ '  
export OS_PROJECT_NAME=finance  
...output omitted...
```

- 14.2. Source the **/home/student/developer5-finance-rc** environment file for the **developer5** user.

```
[student@workstation ~(developer5-finance)]$ source ~/developer5-finance-rc
```

```
[student@workstation ~](developer5-finance)]$
```

- 14.3. Run the **openstack flavor list** command to confirm that the user account is enabled and authentication is working. The command may return no output if no flavor is available.

```
[student@workstation ~](developer5-finance)]$ openstack flavor list
+-----+-----+-----+-----+-----+
| ID      | Name    | RAM | Disk | Ephemeral | VCPUS | Is Public |
+-----+-----+-----+-----+-----+
| 688d...b154 | default | 2048 |   10 |          0 |       2 | True      |
+-----+-----+-----+-----+-----+
```

► 15. Delete the **developer5** user account using the OpenStack CLI.

- 15.1. Source the environment file to provide administrator rights to the **admin** user.

```
[student@workstation ~](developer5-finance)]$ source ~/admin-rc
[student@workstation ~](developer5-finance)]$
```

- 15.2. Delete the **developer5** user account.

```
[student@workstation ~](developer5-finance)]$ openstack user delete developer5
```

- 15.3. Verify that the user has been deleted.

```
[student@workstation ~](developer5-finance)]$ openstack user list
```

Cleanup

On the **workstation** machine, run the **lab administer-users cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab administer-users cleanup
```

This concludes the guided exercise.

ASSIGNING USER ROLES AND PRIVILEGES

OBJECTIVE

After completing this section, students should be able to assign OpenStack user roles and privileges.

INTRODUCTION TO USER ROLES

After a successful authentication, a user is provided access to the OpenStack service based on its authorization. In OpenStack, services verify the roles associated with the user token to provide access to the service. A role defines what actions users can perform upon authentication. Roles that are assigned to assign users to a project are called *user roles*. In OpenStack identity service APIv3, both users and groups can be assigned roles. Roles that are assigned to groups are *group roles*. A group role, granted to a domain or project, applies to all users in the group.

The **admin** user is assigned the **admin** user role for the **admin** project. The **admin** user is similar to the superuser in an operating system, because it has full administrative privileges. Red Hat OpenStack Platform installs with predefined roles, such as the **admin** and **_member_** user roles. The **admin** user role has full privileges to manage almost all resources belonging to any projects.



NOTE

A user assigned the **admin** role in any project currently has **admin** privileges in every project in that domain. Domains in Identity APIv3 were introduced to address the overly high privileges associated with the **admin** user role. Further work is being implemented to define more granular job-specific roles in future releases.

Resource Policy

A cloud administrator can define access policies for each OpenStack service to provide access to resources. For example, a resource can be API access to launch an instance, or create a user. All default resource policies are integrated into the OpenStack code base and loaded on the controller node. Whenever a call to an OpenStack service is made, the service policy engine uses the policy definition to determine if the service request can be accepted.

For example, the **/etc/nova/policy.json** policy file contains an empty brackets as the policies exists in oslo.policy library. On the controller node, use the following command to list the default policies of the Compute service:

```
[heat-admin@controller0 ~]$ sudo oslopolicy-policy-generator --namespace nova
...
① "os_compute_api:servers:create": "rule:admin_or_owner"
...
② "admin_or_owner": "is_admin:True or project_id:%(project_id)s"
...output omitted...
```

- ① Target definition
- ② Rule alias definition

The policy defines a rule alias named **admin_or_owner** that allows it to accept calls from the user with the **admin** user role or users that belong to the same project as that of the resource. The policy definition uses a **target:rule** format. In the previous policy definition, the `os_compute_api:servers:create` API call of the Compute service allows only the users that fulfill the **admin_or_owner** rule alias condition.

The OpenStack policy engine finds the following to determine if access is allowed to the requested resource:

- The policy rule associated with each resource
- The user roles or group roles associated with the user

MANAGING USER ROLES

User roles can be created and edited from the dashboard. As a user with an admin role, log in to the dashboard and navigate to **Identity** → **Roles**. To create a new role, click **Create Role**. To assign a role to a user, choose the role and project while creating the user. Additional user roles can be assigned to an existing user by editing the project members from **Identity** → **Projects**.

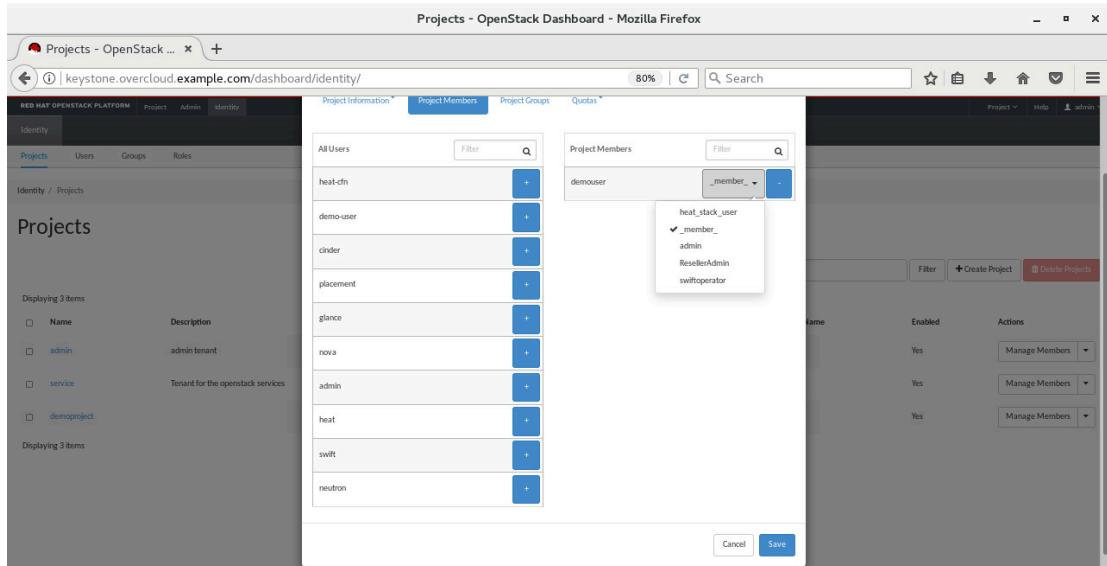


Figure 2.4: Assigning user roles to a user in dashboard

Managing User Roles from the Dashboard

A role defines a set of privileges. When a role is assigned to a user, that user inherits the privileges of that role. When a user requests a service, that service interprets the user role assigned to the user and provides access based on the role. The following steps describe the process of assigning a role to a user from the dashboard.

1. As a user with administrator privileges, navigate to **Identity** → **Projects**.
2. To assign a role to a user, start by selecting the project in which the user is a member. Click **Manage Members** under the **Actions** column.
3. In the Edit project dialog box, click the **Project Members** tab. In the **Project Members** section, select the desired role for the user. Click **Save**.

Managing User Roles from the CLI

With the OpenStack command-line client, use the `openstack role create role-name` command to create the user role. To list the user roles, use the `openstack role list` command.

```
[user@demo ~(admin-admin)]$ openstack role list
+-----+-----+
| ID          | Name      |
+-----+-----+
... output omitted ...
| 9fe2ff9ee4384b1894a90878d3e92bab | _member_      |
| ce3dd5b1e577424fad502ab29324d6d1 | admin        |
... output omitted ...
+-----+-----+
```

Assigning User Roles to Existing Users

To assign a user role to an existing user from the dashboard, edit the project members from Identity → Projects. When using the OpenStack CLI, use the **openstack role add** command to add a role to a user. To add the **demo-role** role to the **demo-user** user in the **demo-project** project, use the following command:

```
[user@demo ~(admin)]$ openstack role add --project demo-project \
--user demo-user demo-role
```

Use the **openstack role assignment list** command to list the user roles assigned to a user in the project.

```
[user@demo ~(admin-admin)]$ openstack role assignment list \
--project demo-project --user demo-user --names
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| Role    | User           | Group | Project           | Domain | Inherited |
|         | demouser@Default |       | demo-project@Default |        | False   |
|         |                 |       |                   |        |          |
| _member_| demouser@Default |       | demo-project@Default |        | False   |
|         |                 |       |                   |        |          |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
```

Viewing User Roles Using the OpenStack CLI

The following steps describe how to use the OpenStack CLI to display the roles assigned to a user.

- As a user with administrator privileges, use the **openstack role list** command to list the user roles. Red Hat OpenStack Platform has some predefined roles like: a **_member_** role that provide self-service access to a project, and an **admin** role that can be assigned to users to administer the OpenStack environment.
- Use the **openstack role assignment list** command to list the roles assigned to a user. Use the **--names** option to display the names of the roles assigned to the user instead of the IDs.

Verifying the Admin User Role

The **admin** user role has admin privileges and has full access rights to most resources in any projects. Users assigned the admin user role can perform almost all tasks in any projects. In the

example below, a user assigned the `_member_` role cannot execute the `openstack user list` command, which requires admin privileges.

```
[user@demo ~](demo-user)]$ openstack user list
You are not authorized to perform the requested action: identity:list_users. ...)
```

The `openstack user list` command, when executed as a user with the `admin` user role, returns all the current users.

```
[user@demo ~](demo-user)]$ openstack user list
+-----+-----+
| ID      | Name   |
+-----+-----+
... output omitted ...
| a6d40a6181bf4cdb8a0ef5bd3393f464 | admin      |
| f95d249a757f47879503c51a305f9534 | demouser  |
... output omitted ...
```



REFERENCES

Additional information may be available in the section on Role Management in the *Users and Identity Management Guide* for Red Hat OpenStack Platform, which can be found at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/users_and_identity_management_guide/

ASSIGNING USER ROLES AND PRIVILEGES

In this exercise, you will use the dashboard and the OpenStack unified CLI to assign user roles to users.

OUTCOMES

You should be able to do:

- Assign administrative privileges to a user.
- Change the user role assigned to a user.
- Verify the assigned user role.
- Understand the difference between the **admin** role and the **_member_** role.

Ensure that **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using the password **student**. On **workstation**, execute the **lab assign-roles** script using the **setup** argument. This script ensures that the **finance** project exists.

```
[student@workstation ~]$ lab assign-roles setup
```

- ▶ 1. On **workstation**, open Firefox and browse to `http://dashboard.overcloud.example.com`.
- ▶ 2. Log in to the dashboard using **admin** as the user name and **redhat** as the password.
- ▶ 3. Create the **architect1** user with **admin** privileges. Set the password for the **architect1** user to **redhat**, and add the user to the **finance** project.
 - 3.1. Navigate to Identity → Users and click +Create User.
 - 3.2. In the dialog box, enter **architect1** in the User Name field, and **redhat** in the Password and Confirm Password fields. Select the **finance** project from the Primary Project list. Select the **admin** role from Role list. Leave the Enabled check box selected and click Create User.
- ▶ 4. Verify that the **architect1** user has **admin** privileges by logging in to the dashboard and creating a user named **developer2**. Set the password for the **developer2** user to **redhat**, and add the user to the **finance** project.
 - 4.1. Log out of the dashboard as the **admin** user and log in as **architect1** using the password **redhat**.
 - 4.2. Navigate to Identity → Users and then click +Create User.

- 4.3. In the dialog box, enter **developer2** in the User Name field, and **redhat** in the Password and Confirm Password fields. Select the finance project from the Primary Project list. Leave Role set to **_member_**, and the Enabled check box selected.

Click Create User.

Because the **architect1** user account has the **admin** user role assigned, the user successfully created the **developer2** account.

- 4.4. Log out of the dashboard.

- 5. Create the **/home/student/architect1-finance-rc** environment file for the **architect1** user.
- 5.1. On **workstation**, open a terminal. Copy the existing **/home/student/admin-rc** file containing the **admin** authentication parameters to **/home/student/architect1-finance-rc**.

```
[student@workstation ~]$ cp ~/admin-rc ~/architect1-finance-rc
```

- 5.2. Edit the **/home/student/architect1-finance-rc** file, and modify the **OS_USERNAME**, **OS_PASSWORD**, **OS_PROJECT_NAME** and **PS1** variables to read as follows:

```
export OS_USERNAME=architect1
export OS_PASSWORD=redhat
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export PS1='[\u@\h \W(architect1-finance)]\$ '
export OS_PROJECT_NAME=finance
...output omitted...
```

- 6. Source the **/home/student/architect1-finance-rc** environment file. Create the **developer3** user with the password **redhat**. Assign the user to the **finance** project. Assign the **admin** user role to the **developer3** user. Verify the roles assigned to the **developer3** user.

- 6.1. Source the **/home/student/architect1-finance-rc** environment file for the **architect1** user.

```
[student@workstation ~]$ source ~/architect1-finance-rc
[student@workstation ~(architect1-finance)]$
```

- 6.2. Create the user **developer3** with the password **redhat** and make it a member of the **finance** project.

```
[student@workstation ~(architect1-finance)]$ openstack user create \
--password redhat \
--project finance \
developer3
+-----+
| Field          | Value           |
+-----+
| default_project_id | 08e3ce2bdb0e46b8834663ba15294934 |
| domain_id      | default          |
| enabled         | True             |
| id              | dcfe3c22da8a4dcfb0123b482a34c05a |
| name            | developer3       |
```

```
| options | {} |  
| password_expires_at | None |  
+-----+-----+
```

- 6.3. Verify that no role is assigned to the **developer3** user in the **finance** project. The command output must not return any value.

```
[student@workstation ~-(architect1-finance)]$ openstack role assignment list \  
--user developer3 \  
--project finance \  
--names
```

- 6.4. Assign the **admin** user role to the **developer3** user.

```
[student@workstation ~-(architect1-finance)]$ openstack role add \  
--project finance \  
--user developer3 \  
admin
```

- 6.5. Verify the user role assigned to the **developer3** user.

```
[student@workstation ~-(architect1-finance)]$ openstack role assignment list \  
--user developer3 \  
--project finance \  
--names  
+-----+-----+-----+-----+  
| Role | User | Group | Project | Domain | Inherited |  
+-----+-----+-----+-----+  
| admin | developer3@Default | | finance@Default | | False |  
+-----+-----+-----+-----+
```

- 7. Create the **/home/student/developer3-finance-rc** environment file for the **developer3** user.

- 7.1. Copy the existing **/home/student/architect1-finance-rc** file containing the **architect1** authentication parameters to **/home/student/developer3-finance-rc**.

```
[student@workstation ~-(architect1-finance)]$ cp ~/architect1-finance-rc \  
~/developer3-finance-rc
```

- 7.2. Edit the **/home/student/developer3-finance-rc** file, and modify the **OS_USERNAME**, **OS_PASSWORD**, and **PS1** variables to read as follows:

```
export OS_USERNAME=developer3  
export OS_PASSWORD=redhat  
export OS_AUTH_URL=http://172.25.250.50:5000/v3  
export PS1='[\u@\h \W(\b\o\ve\l\o\p\er\3-f\in\an\c)e]\$\n'  
export OS_PROJECT_NAME=finance  
...output omitted...
```

- 8. Source the **/home/student/developer3-finance-rc** environment file and delete the **developer2** user.

- 8.1. Source the **/home/student/developer3-finance-rc** environment file for the **developer3** user.

```
[student@workstation ~-(architect1-finance)]$ source ~/developer3-finance-rc  
[student@workstation ~-(developer3-finance)]$
```

- 8.2. Delete the **developer2** user.

```
[student@workstation ~-(developer3-finance)]$ openstack user delete developer2
```

- 9. Delete the **developer3** user. Verify that the user has been deleted.

- 9.1. Delete the **developer3** user.

```
[student@workstation ~-(developer3-finance)]$ openstack user delete developer3
```

- 9.2. Since the **developer3** user has been deleted, the environment variables are insufficient to authenticate a valid user. Attempt to list the OpenStack users.

```
[student@workstation ~-(developer3-finance)]$ openstack user list  
The request you have made requires authentication. (HTTP 401) (Request-ID: req-1e50...b997)
```

- 9.3. Authenticate as a valid user. Source the **/home/student/admin-rc** environment file for the **admin** user.

```
[student@workstation ~-(developer3-finance)]$ source ~/admin-rc  
[student@workstation ~-(developer3-finance)]$
```

- 9.4. Verify that the **developer2** and **developer3** users have been deleted.

```
[student@workstation ~-(developer3-finance)]$ openstack user list \  
--project finance  
+-----+-----+  
| ID | Name |  
+-----+-----+  
| 11941ae6cecc495da6cfce31a0d6f7ed | architect1 |  
...output omitted...  
+-----+-----+
```

- 10. Delete the environment file for the **developer3** user.

```
[student@workstation ~-(developer3-finance)]$ rm -f ~/developer3-finance-rc
```

- 11. Verify that the **developer2** cannot list users in the **finance** project.

- 11.1. Copy the existing **/home/student/architect1-finance-rc** environment file to **/home/student/developer2-finance-rc**.

```
[student@workstation ~-(developer3-finance)]$ cp ~/architect1-finance-rc \
```

- 11.2. Edit the **developer2-finance-rc** environment file, and modify the OS_USERNAME, OS_PASSWORD, and PS1 variables to read as follows:

```
export OS_USERNAME=developer2
export OS_PASSWORD=redhat
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export PS1='[\u@\h \w(\b\w{developer2-finance})]\$ '
export OS_PROJECT_NAME=finance
...output omitted...
```

- 11.3. Run the **openstack user list** command.

```
[student@workstation ~(\b\w{developer3-finance})]\$ source ~/developer2-finance-rc
[student@workstation ~(\b\w{developer2-finance})]\$ openstack user list
The request you have made requires authentication. (HTTP 401) (Request-ID:
req-6867...d789)
```

Cleanup

On **workstation**, run the **lab assign-roles cleanup** script to clean up this exercise.

```
[student@workstation ~]\$ lab assign-roles cleanup
```

This concludes the guided exercise.

MANAGING QUOTAS

OBJECTIVE

After completing this section, students should be able to manage quotas for projects.

INTRODUCTION TO PROJECT QUOTAS

Red Hat OpenStack Platform project quotas are similar to operating system resource quotas. An OpenStack administrator can configure quotas to prevent system resources from being exhausted. Project quotas are set on a per-project basis, and limit the resources that can be assigned. These operational limits enable the cloud administrator to finite control over OpenStack projects. For example, defining a quota based on the amount of RAM allocated to a project. This can prevent clients from committing more memory than necessary, or using more than allowed as defined in their service agreement. Every project has a basic set of quotas with default values. These defaults can be modified when creating new projects and edited for existing projects.

Viewing Default Project Quotas

OpenStack installs with default quota values to provide basic quota restrictions for projects. These values can be modified to provide more resources for project deployments. For example, a project may require more instances to be launched than the OpenStack default quota value of 10.

To view the default quotas from the dashboard, log in as a user with an admin role and navigate to Admin → System → Defaults. The Defaults tab displays the default project quotas.

To view the default quotas from the CLI, run the **openstack quota show --default** command as user with an admin role.

```
[user@demo ~(admin)]$ openstack quota show --default
+-----+-----+
| Field          | Value |
+-----+-----+
| backup-gigabytes | 1000 |
| backups        | 10   |
| cores           | 20   |
| fixed-ips      | -1   |
| floating-ips   | 50   |
| gigabytes       | 1000 |
| groups          | 10   |
| health_monitors | None |
| injected-file-size | 10240 |
| injected-files  | 5    |
| injected-path-size | 255 |
| instances        | 10   |
| key-pairs        | 100  |
| l7_policies      | None |
| listeners         | None |
| load_balancers   | None |
| location          | None |
| name              | None |
| networks          | 100  |
| per-volume-gigabytes | -1   |
```

pools	None	
ports	500	
project	None	
project_name	admin	
properties	128	
ram	51200	
rbac_policies	10	
routers	10	
secgroup-rules	100	
secgroups	10	
server-group-members	10	
server-groups	10	
snapshots	10	
subnet_pools	-1	
subnets	100	
volumes	10	
+-----+-----+		

Updating Default Project Quotas

The default quotas can be updated to set new quota limits for all projects. For the dashboard, log in as a user with an admin role, and navigate to Admin → System → Defaults. Click Update Defaults to update the default project quotas.

An administrator can update the default project quotas using the **openstack quota set** command. For example, run the following command to set the number of instances quota to 15:

```
[user@demo ~(admin)]$ openstack quota set --instances 15 --class default
```

The following table describes the default quotas that are set on a project:

Default Quota Parameters

QUOTA NAME	DESCRIPTION
Server Groups	The number of server groups per project. Server groups can be used to control the affinity and anti-affinity scheduling policy for a group of instances. All instances in a single affinity group are run on the same hypervisor. Instances in an anti-affinity group are run on different hypervisors.
VCPUs	Instance cores (VCPUs) allowed.
Server Group Members	Number of servers in a single server group. Unlimited anti-affinity could allow a user to derive information about the overall size of the cloud, which is generally considered private information of the cloud provider. Unlimited server groups could be used as a DoS attack against systems not protected by an API rate limiter. For instance, a user could create groups until the identity database fills up.
Per Volume Size (GiB)	Maximum volume size in gibibytes. An entry of -1 allows any size.

Quota Name	Description
Injected File Content Bytes	Maximum size, in bytes, of injected file content allowed (per file). The process of putting a file into an instance image before the instance is started is termed file injection.
Length of Injected File Path	Maximum length of the file path for the injected file used for an instance.
Injected Files	Number of injected files allowed per project.
Instances	Number of instances allowed per project.
Key Pairs	Number of key pairs allowed per project.
Metadata Items	Number of metadata items allowed per instance.
RAM (MB)	Megabytes of instance RAM allowed per project.
Volumes	Number of volumes allowed per project.

Creating a Project with Modified Quotas

When projects are created, the project quotas can be defined to override the default values. The quotas can also be changed after creating the project. For example, the number of security groups, security group rules, floating IP addresses, networks, ports, subnets, and routers allowed for the project. Project quotas can be modified from the Dashboard or from the OpenStack CLI.

From the dashboard, navigate to Identity → Projects. Click Create Project to create a project. In the Quotas tab, all fields are populated with default project quota limits. To set non-default quotas for the project, update the values and click Save.

If the quota for number of instances allowed in a project is met, the Launch Instance button in Project → Compute → Instances will be disabled.

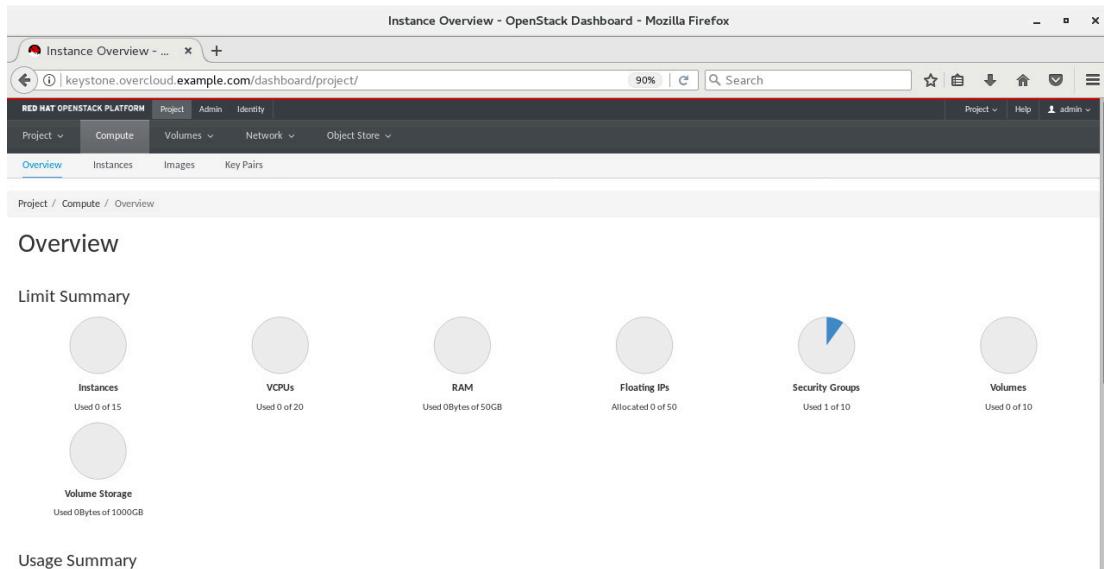


Figure 2.5: Quota Limit Summary in Dashboard

Creating Project with Quotas Using the Dashboard

Red Hat OpenStack Platform installs with default quotas for projects. These project quotas can be edited to define resource usage restrictions on the members of the project. The following steps describe the process for creating a project with default quotas, and then editing these quotas after the project is created.

1. Open the dashboard in a web browser, and log in as a user with an admin role. Navigate to Identity → Projects.
2. Select **Modify Quotas** for a project, under the Actions column. In the Quotas tab, all fields are populated with default project quota limits. For example, the default quota for Instances is 10. This means that only 10 instances can exist in the project at any one time.
Click Save.
3. As an administrator, the default quotas can be viewed and edited by navigating to Admin → System → Defaults. Click Update Defaults to edit the default quotas for all projects. Click again Update Defaults to apply the updated quotas.
4. Log in to the dashboard as an administrator to edit the default quotas for the project. Navigate to Identity → Projects.
5. To edit the project quotas, select **Modify Quotas** in the Actions column. Click Save.
6. Verify the updated quota by logging in to the dashboard as a project user and launching an instance. If any quota is exceeded, it is displayed in red in the Limit Summary section of the Overview page.

After creating the project, the quotas for the project can be set to non-default values. Using the OpenStack command-line client, use the **openstack quota set** command to set the new quota values. For example, to set the **--instances** to 15 in the **demo-project** project use the following command:

```
[user@demo ~](admin)]$ openstack quota set --instances 15 demo-project
```

Similarly, using the command line, an attempt to launch an instance results in the following error.

```
Quota exceeded for instances: Requested 1, but already used 15 of 15 instances  
(HTTP 403) (Request-ID: req-d8cc87d0-3ece-4cca-b0e8-c257d8d202cb)
```

Creating Project with Quotas Using the OpenStack CLI

The following steps outline the process for creating a project with default quotas and editing these quotas after the project is created using the command line.

1. Create and source the an admin user's environment file, and list the default project quotas using the **openstack quota show --default** command.
2. Edit the project quotas using the **openstack quota set** command.
3. Verify the new quota values by listing the project quotas using the **openstack quota show** command.



REFERENCES

Additional information may be available in the section on Quota Management in the *Users and Identity Management Guide* for Red Hat OpenStack Platform, which can be found at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/users_and_identity_management_guide/

MANAGING QUOTAS

In this exercise, you will set, edit, and verify project quotas by launching instances using the dashboard and the OpenStack unified CLI.

OUTCOMES

You should be able to:

- Set project quotas.
- Edit project quotas.
- Verify the project quotas by launching instances.

Ensure that the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using the password **student**. On **workstation**, execute the **lab manage-quotas** script using the **setup** argument. The lab script ensures that the **finance** and **research** projects, **architect1** and **developer2** users, **default** flavor, **rhe17** image, and **finance-network1** private network exist. The lab script also ensures that the environment file for the **architect1** user exists.

```
[student@workstation ~]$ lab manage-quotas setup
```

- ▶ 1. On **workstation**, open Firefox and browse to <http://dashboard.overcloud.example.com>.
- ▶ 2. Log in to the dashboard using **architect1** as the user name and **redhat** as the password.
- ▶ 3. Set the project quotas for the **research** project according to the following table:

QUOTAS CONFIGURATION	
Quotas	2 VCPUs, 2 instances, 2048 MB RAM

- 3.1. Navigate to Identity → Projects. Select Modify Quotas from the list under the Actions column for the **research** project.

Enter **2** for VCPUs, **2** for Instances, and **2048** for RAM (MB).

- 3.2. Click Save.

- ▶ 4. Verify the **RAM(MB)** quota of **2048** MB set on the **research** project. As the **developer2** user, launch two new instances with the **default** flavor. Use the values in the following table when launching the instances:

SETTING	FIRST INSTANCE	SECOND INSTANCE
Instance Name	research-server1	research-server2
Image	rhel7	rhel7
Flavor	default	default
Network (private)	research-network1	research-network1

- 4.1. Log out, and log in as the **developer2** user with the password **redhat**.
 - 4.2. Navigate to Project → Compute → Instances and click Launch Instance.
 - 4.3. In the Details tab, enter **research-server1** in the Instance Name field.
In the Source tab, select Image from the Select Boot Source list. Set the Create New Volume to No. Click ↑ to select **rhel7** as the image.
In the Flavor tab, click ↑ to select **default** as the flavor.
In the Networks tab, ensure that the **research-network1** network has been allocated.
 - 4.4. Click Launch Instance to launch the **research-server1** instance.
 - 4.5. The Launch Instance button is disabled and reads Launch Instance (Quota exceeded), because you reach quota limits. It is not possible to launch **research-server2**.
 - 4.6. Log out of the dashboard.
- 5. Source the **/home/student/developer2-research-rc** environment file and launch the **research-server2** instance using the OpenStack CLI. Use the values in the following table when launching the instance:

SETTING	VALUE
Instance Name	research-server2
Image	rhel7
Flavor	default
Network (private)	research-network1

- 5.1. Source the **/home/student/developer2-research-rc** environment file for the **developer2** user.

```
[student@workstation ~]$ source ~/developer2-research-rc
[student@workstation ~(developer2-research)]$
```

- 5.2. Launch the **research-server2** instance from the command line using the information in the previous table.

```
[student@workstation ~(developer2-research)]$ openstack server create \
--image rhel7 \
--flavor default \
--nic net-id=research-network1 \
--wait research-server2
```

```
Quota exceeded for cores, ram: Requested 2, 2048, but already used 2, 2048 of 2, 2048 cores, ram (HTTP 403) (Request-ID: req-7780...90cca)
```

The instance fails to launch because the VCPUs and RAM requirements exceed the quota for the **research** project.

- ▶ 6. Source the environment file for the **architect1** user to enable admin privileges. Edit the quotas for the **research** project to limit the VCPUs to **4** and RAM to **8192** MB.
 - 6.1. Source the **/home/student/architect1-finance-rc** environment file for the **architect1** user.

```
[student@workstation ~ (developer2-research)]$ source ~/architect1-finance-rc  
[student@workstation ~ (architect1-finance)]$
```

- 6.2. List the current project quotas for the **research** project.

```
[student@workstation ~ (architect1-finance)]$ openstack quota show research  
+-----+-----+  
| Field | Value |  
+-----+-----+  
...output omitted...  
| cores | 2 |  
...output omitted...  
| ram | 2048 |  
...output omitted...  
+-----+-----+
```

- 6.3. Modify the quotas for the **research** project to set the VCPUs to **4**, and the memory to **8192** MB.

```
[student@workstation ~ (architect1-finance)]$ openstack quota set \  
--cores 4 \  
--ram 8192 \  
research
```

- 6.4. List the quotas for the **research** project to confirm the changes made.

```
[student@workstation ~ (architect1-finance)]$ openstack quota show research  
+-----+-----+  
| Field | Value |  
+-----+-----+  
...output omitted...  
| cores | 4 |  
...output omitted...  
| ram | 8192 |  
...output omitted...  
+-----+-----+
```

- 7. Verify that the new quotas are appropriate to launch the **research-server2** instance. Use the **developer2** environment file to launch the **research-server2** instance.

Launch the **research-server2** instance using the **rhel7** image, **default** flavor, and **research-network1** network.

- 7.1. Source the **/home/student/developer2-research-rc** environment file.

```
[student@workstation ~-(architect1-finance)]$ source ~/developer2-research-rc  
[student@workstation ~-(developer2-research)]$
```

- 7.2. Launch the **research-server2** instance, using the **rhel7** image, **default** flavor, and **research-network1** private network.

```
[student@workstation ~-(developer2-research)]$ openstack server create \  
--image rhel7 \  
--flavor default \  
--nic net-id=research-network1 \  
--wait research-server2
```

- 7.3. Verify that the **research-server2** instance was successfully launched.

```
[student@workstation ~-(developer2-research)]$ openstack server list \  
-c Name -c Status  
+-----+-----+  
| Name | Status |  
+-----+-----+  
| research-server2 | ACTIVE |  
| research-server1 | ACTIVE |  
+-----+-----+
```

- 8. Delete the **research-server1** and **research-server2** instances.

```
[student@workstation ~-(developer2-research)]$ openstack server delete \  
research-server1  
[student@workstation ~-(developer2-research)]$ openstack server delete \  
research-server2
```

- 9. Delete the **developer2** user and the **research** project, using the **architect1** authentication parameters.

```
[student@workstation ~-(developer1-finance)]$ source ~/architect1-finance-rc  
[student@workstation ~-(architect1-finance)]$ openstack user delete developer2  
[student@workstation ~-(architect1-finance)]$ openstack project delete research
```

- 10. Delete the identity environment file for the **developer2** user.

```
[student@workstation ~-(architect1-finance)]$ rm ~/developer2-research-rc
```

Cleanup

On **workstation**, run the **lab manage-quotas cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab manage-quotas cleanup
```

This concludes the guided exercise.

ORGANIZING PEOPLE AND RESOURCES

PERFORMANCE CHECKLIST

In this lab, you will manage a project and its users, and set quotas for the project. You can perform the tasks in this lab using either the dashboard or the OpenStack unified CLI.

OUTCOMES

You should be able to:

- Create and delete a project.
- Create, delete, and add a user to a project.
- Edit and enable a user account.
- Set project quotas.

Ensure that the **workstation** and overcloud's virtual machines are started. Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run **lab resources-lab setup**, which creates the **production** and **marketing** projects. The lab script also creates the **operator1**, **operator2** and **operator3** users, and disables the **operator2** user.

```
[student@workstation ~]$ lab resources-lab setup
```

1. To perform the lab tasks on **workstation** using the dashboard, open Firefox and browse to <http://dashboard.overcloud.example.com>.
To perform the lab tasks on **workstation** using the OpenStack unified CLI, open a terminal session.
2. Log in to the dashboard using **architect1** as the user name and **redhat** as the password.
Alternatively, to use the OpenStack unified CLI, source the **/home/student/architect1-production-rc** file containing the **architect1** authentication parameters.
3. Create a project named **consulting**.
4. Create a user named **operator4** with **redhat** as the password. The **operator4** user must be a unprivileged member of the **consulting** project.
5. Create a user named **architect2** with the **admin** role for the **consulting** project. Set **redhat** as the password for the user.
6. Set the project quotas for the **consulting** project to have 4 VCPUs, 7500 MB RAM, and 4 instances.
7. Enable the **operator2** user.
8. Change the email address for the **operator2** user to **student@workstation.lab.example.com** and the password to **redhat**.
9. Delete the **operator3** user account.

10. As the **architect1** user, delete the **marketing** project and log out from the dashboard.

Evaluation

On **workstation**, run the **lab resources-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab resources-lab grade
```

Cleanup

On **workstation**, run the **lab resources-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab resources-lab cleanup
```

This concludes the lab.

ORGANIZING PEOPLE AND RESOURCES

PERFORMANCE CHECKLIST

In this lab, you will manage a project and its users, and set quotas for the project. You can perform the tasks in this lab using either the dashboard or the OpenStack unified CLI.

OUTCOMES

You should be able to:

- Create and delete a project.
- Create, delete, and add a user to a project.
- Edit and enable a user account.
- Set project quotas.

Ensure that the **workstation** and overcloud's virtual machines are started. Log in to **workstation** as **student** using **student** as the password.

On **workstation**, run **lab resources-lab setup**, which creates the **production** and **marketing** projects. The lab script also creates the **operator1**, **operator2** and **operator3** users, and disables the **operator2** user.

```
[student@workstation ~]$ lab resources-lab setup
```

1. To perform the lab tasks on **workstation** using the dashboard, open Firefox and browse to <http://dashboard.overcloud.example.com>.
To perform the lab tasks on **workstation** using the OpenStack unified CLI, open a terminal session.
2. Log in to the dashboard using **architect1** as the user name and **redhat** as the password.
Alternatively, to use the OpenStack unified CLI, source the **/home/student/architect1-production-rc** file containing the **architect1** authentication parameters.
 - 2.1. For the dashboard, enter **architect1** as the user name and **redhat** as the password on the dashboard.
 - 2.2. To perform the task from the command line, source the **/home/student/architect1-production-rc** file containing **architect1** authentication parameters. The file sets the OpenStack identity authentication endpoint, user name, password, region, and project for the OpenStack command-line clients.

```
[student@workstation ~]$ source ~/architect1-production-rc  
[student@workstation ~(architect1-production)]$
```

3. Create a project named **consulting**.
 - 3.1. Navigate to Identity → Projects and click +Create Project.

Enter **consulting** in the Name field and **consulting** in the Description field. Leave the Enabled check box selected. Click Create Project.

- 3.2. Alternatively, create the **consulting** project from the command line.

```
[student@workstation ~(architect1-production)]$ openstack project create \
--description consulting \
--enable consulting
+-----+-----+
| Field      | Value           |
+-----+-----+
| description | consulting      |
| domain_id   | default          |
| enabled     | True             |
| id          | 744b3acea749493588ece8655b9ae477 |
| is_domain   | False            |
| name        | consulting       |
| parent_id   | default          |
| tags        | []               |
+-----+-----+
```

4. Create a user named **operator4** with **redhat** as the password. The **operator4** user must be a unprivileged member of the **consulting** project.

- 4.1. On the Identity tab, select the Users tab and click +Create User.

Enter **operator4** in the User Name field. Enter **redhat** in the Password and Confirm Password fields. Select the consulting project from the Primary Project list. Ensure that **_member_** is selected from the Role list. Leave the Enabled check box selected. Click Create User.

- 4.2. Alternatively, create the **operator4** user from the command line.

```
[student@workstation ~(architect1-production)]$ openstack user create \
--project consulting \
--password redhat \
operator4
+-----+-----+
| Field      | Value           |
+-----+-----+
| default_project_id | 744b3acea749493588ece8655b9ae477 |
| domain_id   | default          |
| enabled     | True             |
| id          | 368e8a021ed54e02b8ec4e8687ab4cd1 |
| name        | operator4       |
| options     | {}               |
| password_expires_at | None            |
+-----+-----+
```

- 4.3. Assign the **_member_** role to the **operator4** user.

```
[student@workstation ~(architect1-production)]$ openstack role add \
--project consulting \
--user operator4 \
_member_
```

5. Create a user named **architect2** with the **admin** role for the **consulting** project. Set **redhat** as the password for the user.
- 5.1. In the Users tab, click Create User.

Enter **architect2** in the User Name field. Enter **redhat** in the Password and Confirm Password fields. Select the consulting project from the Primary Project list. Select the **admin** role from the Role list. Leave the Enabled check box selected. Click Create User.
 - 5.2. Alternatively, use the command line to create the **architect2** user as a member of the **consulting** project. Set **redhat** as the password for the user.

```
[student@workstation ~-(architect1-production)]$ openstack user create \
--project consulting \
--password redhat \
architect2
+-----+
| Field          | Value           |
+-----+
| default_project_id | 744b3acea749493588ece8655b9ae477 |
| domain_id      | default          |
| enabled         | True             |
| id              | 49a3becce0224487bf606f87af637c26 |
| name            | architect2       |
| options          | {}               |
| password_expires_at | None             |
+-----+
```

- 5.3. Assign the **admin** role to the **architect2** user.

```
[student@workstation ~-(architect1-production)]$ openstack role add \
--project consulting \
--user architect2 \
admin
```

6. Set the project quotas for the **consulting** project to have 4 VCPUs, 7500 MB RAM, and 4 instances.
- 6.1. Navigate to the Identity → Projects tab. Select Modify Quotas from the list under the Actions column for the **consulting** project.
 - 6.2. Change the quota value for VCPUs to **4**, Instances to **4**, and RAM (MB) to **7500**, and then click Save.
 - 6.3. Alternatively, when using the OpenStack CLI, set quotas for the **consulting** project.

```
[student@workstation ~-(architect1-production)]$ openstack quota set \
--cores 4 \
--instances 4 \
--ram 7500 \
consulting
```

7. Enable the **operator2** user.
 - 7.1. Navigate to Identity → Users. Select Enable User from the list under the Actions column for the user **operator2**.
 - 7.2. Alternatively, enable the **operator2** user using the command line.

```
[student@workstation ~(architect1-production)]$ openstack user set \
--enable operator2
```

8. Change the email address for the **operator2** user to **student@workstation.lab.example.com** and the password to **redhat**.
 - 8.1. On the Users tab, select Edit in the Actions column for the **operator2** user. Enter **student@workstation.lab.example.com** in the Email field. Click Update User. Change the password for the **operator2** user. Select Change Password under the Actions column. Enter **redhat** for the new password and click Save.
 - 8.2. Alternatively, change the email address and password for the **operator2** user using the command line.

```
[student@workstation ~(architect1-production)]$ openstack user set \
--email student@workstation.lab.example.com \
--password redhat \
operator2
```

9. Delete the **operator3** user account.
 - 9.1. On the Users tab, select the **operator3** user. Click Delete Users. Click Delete Users again to confirm.
 - 9.2. To perform this step using the OpenStack CLI, use the **openstack user delete** command to delete the **operator3** user.

```
[student@workstation ~(architect1-production)]$ openstack user delete operator3
```

10. As the **architect1** user, delete the **marketing** project and log out from the dashboard.
 - 10.1. Navigate to Identity → Projects. Select the **marketing** project. Click Delete Project under the Action column. Click Delete Project again to confirm. Log out of the dashboard.
 - 10.2. Alternatively, to delete the **marketing** project using the command line, use the **project delete** subcommand to the **openstack** command.

```
[student@workstation ~(architect1-production)]$ openstack project delete \
marketing
```

Evaluation

On **workstation**, run the **lab_resources-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab_resources-lab grade
```

Cleanup

On **workstation**, run the **lab_resources-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab resources-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- In OpenStack, the authentication and authorization services are provided by the Identity Service.
- Red Hat OpenStack Platform supports Identity service APIv2 and APIv3. APIv3 introduces new features such as domains and groups.
- Domains are collections of projects and users that can be defined within a single identity domain.
- The Identity Service supports organizational mapping for authentication purposes using several elements: domains, users, and projects.
- Authorization to a particular OpenStack service is based on roles and resource policies.
- A user can be a member of one or more projects, several users can be members of a single project, and a user can have different roles in different projects.
- The **admin** user role has full privileges across all the projects.
- Project quotas are set on a per-project basis and limit the amount of resources that can be assigned.

DESCRIBING CLOUD COMPUTING

GOAL

Describe the changes in technology and processes for cloud computing.

OBJECTIVES

- Describe cloud computing concepts.
- Describe virtual machines and containers.
- Illustrate use cases for Red Hat Enterprise Linux, Red Hat CloudForms, Red Hat Virtualization, and Red Hat OpenStack Platform.
- Verify OpenStack services.

SECTIONS

- Describing Cloud Computing Concepts (and Quiz)
- Describing Virtual Machines and Containers (and Quiz)
- Illustrating Red Hat Cloud Product Use Cases (and Quiz)
- Verifying OpenStack Services (and Guided Exercise)

QUIZ

- Describing Cloud Computing

DESCRIBING CLOUD COMPUTING CONCEPTS

OBJECTIVES

After completing this section, students should be able to describe cloud computing concepts.

WHAT IS CLOUD COMPUTING?

Cloud computing is a model for ubiquitous, convenient, on-demand network access to a shared pool of configurable computing resources, such as servers, storage, networks, applications, and services. Resources can be rapidly provisioned with minimal management or service provider interaction. Self-service users launch *instances* as virtual machines, when needed, without requiring assistance from the service provider. The *cloud consumer* decides the computing resources needed for their instances. Cloud computing has several core characteristics:

- *Self-service*: Allows cloud consumers to provision instances with computing resources.
- *Multitenancy*: Allows multiple cloud consumers to share underlying hardware and resources.
- *Elasticity*: Dynamically increases or decreases the resources used according to demand.
- *Telemetry*: Resources can be monitored and metered by the service provider and the consumer.

Cloud Types

Cloud computing is available in three forms: public, private, and hybrid clouds, offering different levels of security, management, and accessibility.

- *Public*: Originally started as application hosting services, but has evolved to be applications, infrastructure, and data storage served by a third party vendor.
- *Private*: On-premises data centers that include virtualization, software, and automation to build and manage infrastructure, feature-matching the public cloud, providing greater control over security, data privacy, and compliance, plus the flexibility of public cloud tools and capabilities.
- *Hybrid*: Configuring both public and private cloud resources to be orchestrated together. Use cases include client facing apps on the public cloud connected to data storage in private cloud datacenters, or building apps that run on identical software infrastructure on both private and public, while responding to changes in demand by distributing capacity. Hybrid designs using OpenStack are being proven to be the highest flexibility and adaptability.

Cloud Models

There are three models for cloud service offerings, based on the service provided:

- *Infrastructure as a Service* (IaaS) allows the cloud consumer to provision computing resources and software (operating system and applications). The cloud consumer manages the instance images, storage, networking, and computing resources (vCPUs and vRAM).
- *Platform as a Service* (PaaS) provides the operating system (i.e., libraries, programming languages) for the cloud consumer. The cloud consumer provides the application(s) to be deployed on the instances. The provider manages the underlying cloud infrastructure.
- *Software as a Service* (SaaS) provides the operating system and all software for the cloud consumer. The provider manages the underlying cloud infrastructure and the application(s) deployed for use as live services.

TRADITIONAL WORKLOADS VERSUS CLOUD WORKLOADS

Cloud environment capabilities are evolving at a dizzying pace. Even in the few years before this course release, many assumptions and limitations that previously defined traditional and cloud differences are now discarded. The explanations presented here are basic, and are not intended to represent all history, implementations or solutions.

Traditional workloads

Traditional workload describes an application designed using an enterprise computing model prevalent before the introduction of cloud. Such applications today are described as *legacy* workloads. Structurally, enterprise applications use a *tightly-coupled* programming model, designed for high speed and optimal latency. Traditional workloads could be client-server or multi-tier middleware designs.

Service Oriented Architecture (SOA) is considered as the culmination of traditional design, in which a large enterprise application is architected as service components, providing service to other components using network protocols. Each service is a discrete functional unit running on a server or as a virtual machine, designed to offer a specific and predictable business result or activity. SOA is commonly misinterpreted as being a cloud design, because some enterprise applications achieved advanced service distribution, high availability and uptime reliability. SOA is a forerunner to cloud design; as developers realized that a computing environment expecting applications to programmatically handle complex scaling was not sustainable.



Figure 3.1: Traditional workloads scale up to larger monolithic systems

Traditional workloads, including SOA, are defined as *monolithic*, meaning that each service or component is discrete, handling higher loads by being programmed to grow larger. This is why traditional workloads are described as *scaling up*, being moved to larger and larger systems as the applications or their respective components evolved. Enterprise virtualization is also an example of traditional workloads, adding more RAM, CPUs, and storage as workloads increase. However, even the largest systems have upper limits.

Traditional enterprise applications can be made to be resilient and scalable, but such features must be implemented through additional high availability management tools, and with the programmatic cooperation of the the applications themselves. Complex applications have unique needs, such that each enterprise application usually required custom programming to implement scaling. The difficulty in creating industry standard solutions for monolithic applications is what led to the community effort to redesign enterprise computing with scaling, high availability and resiliency an integral part of the design.

Cloud workloads

Cloud workload describes an application or service offering design to take advantage of the benefits of an Infrastructure-as-a-Service environment, such as on-demand scaling, distributed access, and a realistic ability to achieve 100% uptime, excluding unprecedeted events. Cloud applications, like SOA, are designed as a stack of service or microservice components, each deployed and managed as separate components. Service components deploy as VMs or containers and scale by starting numerous instances of each service.

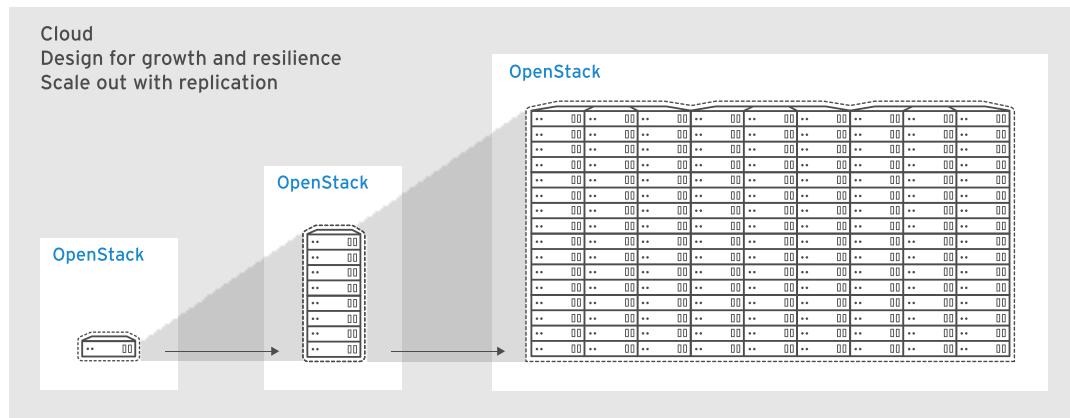


Figure 3.2: Cloud workloads scale out to replicate and load-manage service instances

Cloud applications are designed different than enterprise applications. For example, enterprise applications were programmatically taught how to scale and process work as they scaled larger, with hardware limitations and bottlenecks driving program solutions and design. Cloud application scale, but the application or service component has no programming or understand that it is being scaled. Service components are replicated and managed by load balanced networking; scaling occurs by starting as many copies of a service as is required to handle the current load. However, in order to be scaled, a service must follow standard cloud design rules:

- Create a service at an optimal, efficient size, for duplication and tuning.
- Use resource structures that do not bottleneck, i.e., avoid mutex file-based parameters.
- Use cloud-design caching to share data, such as memcached or Redis.
- Use loosely coupled designs that easily tolerate individual service failures.
- Use non-proprietary, portable features.
- Use cloud-based message broker communication between components.

Cloud design is still rapidly evolving. Only a few years back, recommended practice stated that some application types should not be moved to the cloud; such as proprietary databases, non-ephemeral workload processing, workloads requiring massive, low latency networking, and applications with unique or dedicated hardware needs.

However, numerous solutions have appeared and continue to evolve. Terabyte-sized SSDs as ephemeral disks allow databases to reside in deployed instances, clustered and shared using open source software such as MySQL, Galera, Redis and memcached. Network speed and latency approach bare metal speeds using technologies allowing cloud instances to directly access virtualized, physical interface channels on host servers. Within realistic limitations of today's business and telecom solutions, any workload can be deployed and optimized on the cloud. Only high performance computing itself, which uses massively specialized equipment, does not gain from the benefits or features of today's cloud environments.

Traditional network administrators configure and manage a stationary networking architecture consisting of physical hardware such as network routers and switches.

Cloud network administrators expand these responsibilities to include a dynamic architecture consisting of virtual devices managed and configured in software. *Software defined networking*

(SDN), allows cloud network administrators to programmatically configure and manage virtual networking devices such as switches, bridges, routers, segments, VLANs and network interfaces.

Other organizational user roles such as cloud operator, domain operator, or additional cloud admins associated with a particular project are common. Each of these users can be granted access to services based on their role and the attributes set by the admin user in the services **policy.json** file.

BENEFITS OF OPENSTACK

Red Hat OpenStack Platform permits heterogeneous environments supporting cloud deployments to integrate with the open standards-based OpenStack technology, all with the flexibility for future changes. As innovation is made within the OpenStack community and released by the OpenStack Foundation, Red Hat will ensure its platform maintains full interoperability to ensure ongoing integration of all cloud infrastructure components in an environment. Perpetual interoperability coupled to the broad vendor ecosystem ensures companies that adopt Red Hat OpenStack Platform can continue to purchase low cost commodity hardware to meet demand and replace end of life equipment. With Red Hat OpenStack Platform, customers are not subject to cost and availability pitfalls that can happen within a proprietary approach. Some of the benefits of using OpenStack for building private clouds are:

Standardization in Its Foundation

The promise of the cloud has greatly eliminated the days when large technology providers tried to lock in customers with monolithic closed systems. But that promise is dependent upon standardization. Thus, more than 700 companies supporting OpenStack are striving toward a flexible, standardized platform that works interchangeably with any infrastructure. This is extremely important, especially since many companies have spent years investing large sums of money in IT.

Less Cost and More Innovation

Most IT departments are focused on running and managing the infrastructure and not providing innovative solutions. The flexibility and low cost of OpenStack helps alleviate this by freeing up IT to focus on new applications, solutions, and service delivery rather than inflexible, underlying infrastructure. This allows for faster delivery of new features and products, such as online tools to help customers better manage their portfolios, and can help attract customers and increase retention.

Industry-wide Support

OpenStack receives widespread support from some of the most important corporations in the technology industry, all of which have come together to help companies break away from being locked in to a particular cloud vendor. While some of these players offer their own flavor of OpenStack, they still commit to the ideals of an open, standardized cloud. The major concern is not choosing the right technology, but selecting the vendor with the richest ecosystem and support, knowing that support extends to virtually an entire industry.

Enables Portability to Other Clouds

Investments in open cloud development like OpenStack must be portable to other clouds. Portability takes a variety of forms, including programming languages and frameworks, data, and applications. If developing an application for one cloud, it should not need to be rewritten in a different language or use different APIs to move it somewhere else.

Deployable on the Infrastructure of Choice

OpenStack provides an additional layer of abstraction above virtualization, physical servers, storage, networking, and public cloud providers. This requires that cloud management be

independent from specific virtualization and other infrastructure technologies. OpenStack can work on **libvirt**, **kvm**, or **qemu**.



REFERENCES

Further information is available in the Chapter 1: Understanding Red Hat OpenStack Platform section of the *Product Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/product_guide/

DESCRIBING CLOUD COMPUTING CONCEPTS

Choose the correct answers to the following questions:

► 1. What are the advantages of using cloud deployments over traditional data centers?

(Choose one.)

- a. Scalable, centralized control, licensing model tied to hardware, abstraction of hardware from user interface
- b. Scalable, self-service, licensing model tied to hardware, infrastructure-agnostic
- c. Scalable, self-service, licensing model not tied to hardware, infrastructure-agnostic
- d. Stateless, scale up only, self-service, infrastructure-agnostic

► 2. List two benefits of OpenStack over other private cloud providers. (Choose two.)

- a. Enables portability to other clouds
- b. Industry-wide support
- c. Open Source software
- d. Deployable only on certain hardware
- e. Can only be implemented using KVM virtualization platform

► 3. Which cloud model does Red Hat OpenStack Platform belong to?

- a. PaaS
- b. SaaS
- c. IaaS
- d. LBaaS

► 4. Which three cloud computing forms best describe the cloud types deployed with Red Hat OpenStack Platform? (Choose three.)

- a. Hybrid
- b. Open
- c. Public
- d. Private
- e. Service

DESCRIBING CLOUD COMPUTING CONCEPTS

Choose the correct answers to the following questions:

► 1. What are the advantages of using cloud deployments over traditional data centers?

(Choose one.)

- a. Scalable, centralized control, licensing model tied to hardware, abstraction of hardware from user interface
- b. Scalable, self-service, licensing model tied to hardware, infrastructure-agnostic
- c. Scalable, self-service, licensing model not tied to hardware, infrastructure-agnostic
- d. Stateless, scale up only, self-service, infrastructure-agnostic

► 2. List two benefits of OpenStack over other private cloud providers. (Choose two.)

- a. Enables portability to other clouds
- b. Industry-wide support
- c. Open Source software
- d. Deployable only on certain hardware
- e. Can only be implemented using KVM virtualization platform

► 3. Which cloud model does Red Hat OpenStack Platform belong to?

- a. PaaS
- b. SaaS
- c. IaaS
- d. LBaaS

► 4. Which three cloud computing forms best describe the cloud types deployed with Red Hat OpenStack Platform? (Choose three.)

- a. Hybrid
- b. Open
- c. Public
- d. Private
- e. Service

DESCRIBING VIRTUAL MACHINES AND CONTAINERS

OBJECTIVES

After completing this section, students should be able to describe virtual machines and containers.

WHAT IS A VIRTUAL MACHINE?

Virtualization allows a single physical computer to host and support multiple operating systems or application environments, each as a software emulation of a complete computer system, known as a virtual machine. Each virtual machine runs its own operating system, which can be different from the one on the host computer. The host system runs specialized software known as a hypervisor, to emulate BIOS, CPUs, memory, network devices and other physical resources for each virtual machine. The host's operating system uses a technique known as namespaces to securely isolate each virtual machine. An end user on a virtual machine has the same experience as on a physical system. Virtual machines are themselves unaware that their resources are virtual, as they appear to have their own hardware devices, interfaces, file systems and software applications.

Virtualization is accomplished by using a hypervisor. A hypervisor is the software that manages and supports the virtualization environment. It runs the virtual machines for each virtualized operating system, providing access to virtual CPUs, memory, disks, networking, and other peripherals while restricting the virtual machines from having direct physical access to the other virtual machines. The hypervisor used in Red Hat Enterprise Linux is *KVM* (Kernel-based Virtual Machine). With adequate hardware, a single hypervisor can host a large number of virtual machines; sharing the physical resources more efficiently than the legacy enterprise model where each operating system or application environment required a physical system.

WHAT IS A CONTAINER?

A container is an isolated emulation environment similar to a virtual machine. Like a virtual machine, it behaves as a separate computer system, using hardware emulated in software by the host system. The difference is that a container is designed to share the host's operating system, and provide only the additional libraries and configuration files required to support the application that it contains. Because containers normally do not contain a full operating system, they are much smaller and faster to start and scale.

In current cloud and hybrid application design, almost all applications and services are offered as web applications using the HTTP protocol through generic and highly evolved web service frameworks, which are dependent on any one specific operating system. Consequently, Linux has become the commonly used operating system for templates and images in cloud environments. It is no longer necessary to run virtual machines with different operating systems, it is only necessary to have similar namespace isolation on top of a shared operating system. Container technology allows each isolated instance to override libraries or files from the host operating system to use modified libraries and files required exclusively for that container's application.

The container framework creates an emulated execution environment in which applications do not actually have direct physical access to files, devices, sockets, or processes outside of the container. If the application requires access to resources that are external to the container, a network connection is opened, acting as if the container was a separate physical server.

Each container has an exclusive virtual network interface and private IP address allowing it to access the outside world. Containers provide a way to deploy an application and its dependencies as self-contained units, without interference and conflicts caused by other applications (and their

dependencies) that are sharing the same host OS. A container only needs the subset of resources directly required by the application, such as shared libraries, user commands, and runtime services.

COMPARING VIRTUAL MACHINES AND CONTAINERS

Virtual machines and containers each have design differences that determine their optimal use cases and deployment environments.

Comparing virtual machines and containers

VIRTUAL MACHINE	CONTAINER
A VM has its own operating system that uses hardware virtual machine support provided by a host hypervisor.	A container runs on a container management system utilizing the operating system on which it is installed to provide namespace isolation and basic operating system services for all container applications. No hardware-emulating hypervisor is required.
VMs are larger because their memory footprint includes most or all of an operating system.	Container systems have a lower overhead because they contain only the application and libraries. Container systems typically target environments where thousands of containers are in play.
VMs typically exist in environment where the number of instances is measured in hundreds, since most are enterprise applications with a monolithic core design.	Containers are typically deployed in environments where thousands of containers exist simultaneously. The difference is that container applications are implemented as numerous microservices.
VMs use emulated drivers that generally provide the same services inside the virtual machines as provided on the host system. I/O virtualization can provide direct hardware access to a virtual machine.	Container management systems provide service isolation between containers, such that direct container services such as file systems and networking have limited resource access.
VMs are abstract systems which use device drivers to service abstract virtual machines. Para-virtualized VM environments provide a hardware abstraction layer (HAL) requiring HAL-specific device drivers.	A container abstracts the whole operating system relying on the container management system for all service interpretation.
A VM can run many applications which change and are reconfigured, similar to legacy systems.	A container is normally one application or a tightly related set of applications, as a service.
VMs systems can run different operating systems.	Containerized applications run on a container manager on a supported operating system. Operating systems have evolved to include container management; containers run natively on operating systems such as Red Hat Enterprise Linux and CoreOS.

VIRTUAL MACHINE	CONTAINER
Virtual machine technology has been established for a long time in enterprise computing. Less investment is being made in hypervisor virtualization design.	Containers have also been established for decades, but only new enhancements in delivery and management have resulted in common use today. Companies are investing heavily in advancing container technologies.
VM boot times are longer because they load an operating system, instantiate virtual device instances, and load and manage the space required for a VM's disk.	Containers have a much smaller overhead, with faster start times since they load only the needed application binaries and libraries.

Virtual machines and containers can coexist. Although containers are designed to all share the host operating system, containers and container management systems can be installed into virtual machines. Red Hat OpenStack Platform and Red Hat Enterprise Linux are designed to facilitate mixed technology support, so that a diverse set of container requirements can be run on a single container platform.

Virtual Machines and Containers

The following figure compares the composition of virtual machines on a virtualization host with the composition of containers on a host operating system :

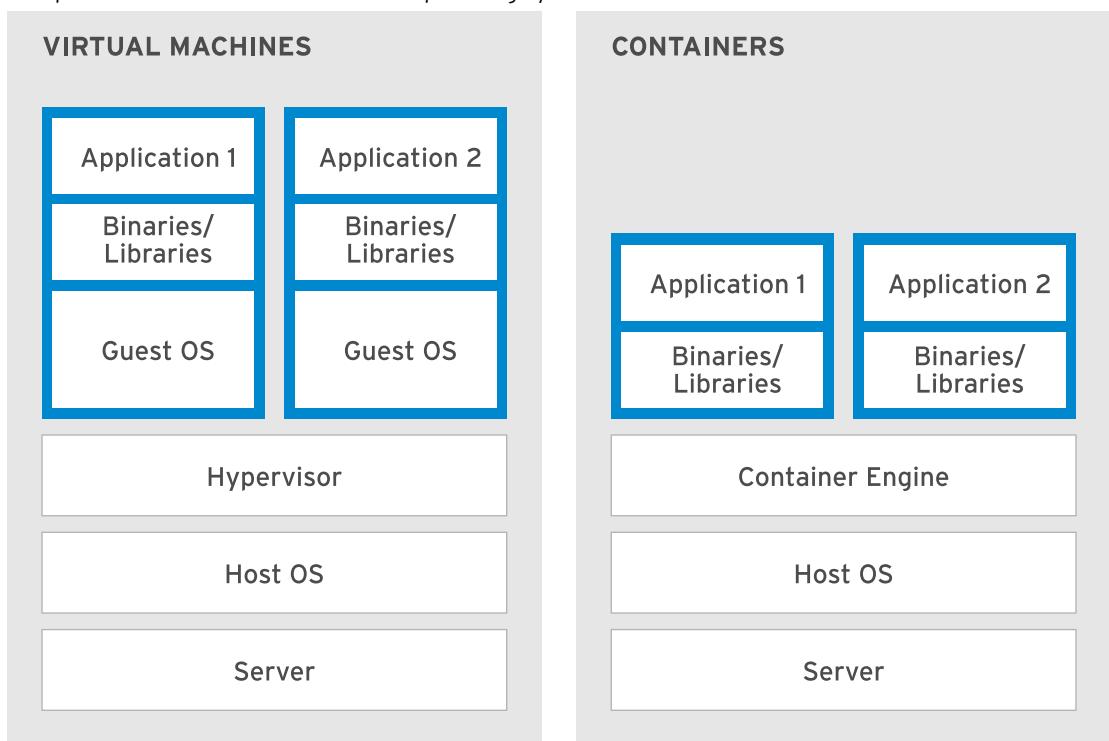


Figure 3.3: Virtual Machines and Containers

About virtual machines:

- KVM enables virtual machines to boot any operating system, including non-Linux systems.
- Running discrete kernel instances provides process separation, which boosts security. The unexpected termination of a virtual machine kernel does not impact other virtual machines.

About containers:

- Containers are designed to support isolation of one or more applications.
- A running container uses a file system image containing the application process and its dependencies including shared libraries and binaries.
- System-wide changes are visible in each container. For example, a host system kernel upgrade would apply to all containers.
- Since containers are relatively lightweight, a large number can run simultaneously on a host system. The theoretical maximum is 6,000 containers and 12,000 bind mounts of root file system directories.

Virtual Machine Use Cases

Consolidation

Multiple physical servers converted into virtual machines requires fewer physical hosts.

Modern data centers are reaching limits that make expansion challenging. Acquiring sufficient physical resources is expensive. More servers require more power and cooling.

Underutilized servers have become commonplace because modern hardware is powerful, and data centers often use separate servers to segregate services for security reasons. Services may have conflicting version requirements in the software, libraries, or operating systems, mandating separate systems for each configuration.

Virtualization helps solve this problem by consolidating underutilized physical servers into virtual servers on fewer systems. Consolidation lowers data center power consumption and heat generation and creates more rack space for expansion and other purposes.

Compartmentalization

Virtualization is used to divide existing physical machines into separate virtual machines without additional hardware expenditures.

For example, a server running several critical services can be divided into multiple virtual machines, each running one critical service. An attacker might compromise one virtual machine, but if the breach does not provide management access to the virtualization software, other domains remain secure. Single physical servers are still a single point of failure, but other implemented technologies, such as live migration, can solve that issue.

Users can be provided full administrative rights on a typical, unprivileged domain for their server applications, while the IT group retains tighter control of locked-down privileged domains used to manage systems. Examples include taking control of user domains for mandatory updates and auditing purposes.

Development and Testing

Virtualization offers advantages for software and network service development and testing. On a single machine, a developer could run several different test environments. If a test environment crashes, it does not corrupt the developer's main working environment.

Reinstalling test environments to a known state can be rapid, using archived disk images and logical volume snapshots, minimizing down time due to rebuilds. Developers can collect crash dumps from failed virtual machines using tools on privileged domains, and analyze dumps using tools set up in advance.

Virtual Appliances

Virtual appliances are preconfigured virtual machines running useful applications in an easily deployed manner. Software vendor consider this to be valuable, since it is easier to ship a preinstalled system with a known configuration, versioning, and environment tuning, instead of software packages to be installed and configured onsite.

Container Use Cases

Faster Delivery

Containers facilitate faster delivery of applications. Developers can write applications on local containers, then publish to image registries for consumption by other teams, such as quality control or production.

Easier Deployment and Scaling of Applications

Containers allow for easier deployment, are more portable and scale better. Containers can be created on a developer's local host, and can run without modification on another physical machine or as a virtual guest. They can run in the office, in the data center, or in a cloud.

Higher Density

Containers achieve higher density. They are lightweight, fast, and a cost-effective alternative to hypervisor-based systems, especially useful in high-density computing environments.

OPENSTACK CONTAINER DEPLOYMENT TOOLS (KOLLA)

A typical OpenStack installation consists of many internal services running on multiple machines. These services have traditionally been Linux system services that can be difficult to deploy, maintain and operate as redundant, load-balanced, resilient services. Until the introduction of containers in the Infrastructure as a Service (IaaS) level.

The Containerization Deployment tools (project name **kolla**) simplifies the deployment, configuration and management of OpenStack by running services inside Docker containers. The deployment tools use **paunch**, a utility to launch and manage containers using YAML based configuration data.

Red Hat builds, provides and maintains the OpenStack services container images, and director installs them as containers during tripleO deployment. All core OpenStack services and most of the included supplemental services are containerized in the current release. Some services, such as Galera, Pacemaker, and message brokers, provide their own high availability architecture and may not be candidates for containerization.



REFERENCES

Further information is available in the Introduction to Linux Containers section of the *Overview of Containers in Red Hat Systems* for Red Hat Enterprise Linux Atomic Host at

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_atomic_host/7/html-single/overview_of_containers_in_red_hat_systems/index

DESCRIBING VIRTUAL MACHINES AND CONTAINERS

Choose the correct answers to the following questions:

► 1. Which statement is true about virtual machines?

- a. Virtual machines cannot be isolated from each other when running on the same virtualization host.
- b. By default, all physical hardware on the virtualization host can be directly accessed by any virtual machine running on the same virtualization host.
- c. Virtual machines run in a virtualization environment managed by a hypervisor.
- d. A hypervisor is only necessary if there is more than one virtual machine running on the virtualization host.

► 2. From within an execution environment, by default, which item allows access to the outside world?

- a. Public Files
- b. Host Device
- c. IP Address
- d. Processes

► 3. Which three statements are true about containers? (Choose three.)

- a. Kernel upgrades are applied to all the running containers.
- b. Given the same hardware configuration, more containers than virtual machines can run on the same host.
- c. Applications running in containers require a dedicated kernel to run within the container in addition to the application process and all its dependencies.
- d. Containers support isolation of one or more applications.

DESCRIBING VIRTUAL MACHINES AND CONTAINERS

Choose the correct answers to the following questions:

► 1. Which statement is true about virtual machines?

- a. Virtual machines cannot be isolated from each other when running on the same virtualization host.
- b. By default, all physical hardware on the virtualization host can be directly accessed by any virtual machine running on the same virtualization host.
- c. Virtual machines run in a virtualization environment managed by a hypervisor.
- d. A hypervisor is only necessary if there is more than one virtual machine running on the virtualization host.

► 2. From within an execution environment, by default, which item allows access to the outside world?

- a. Public Files
- b. Host Device
- c. IP Address
- d. Processes

► 3. Which three statements are true about containers? (Choose three.)

- a. Kernel upgrades are applied to all the running containers.
- b. Given the same hardware configuration, more containers than virtual machines can run on the same host.
- c. Applications running in containers require a dedicated kernel to run within the container in addition to the application process and all its dependencies.
- d. Containers support isolation of one or more applications.

ILLUSTRATING RED HAT CLOUD PRODUCT USE CASES

OBJECTIVES

After completing this section, students should be able to identify use cases for Red Hat Enterprise Linux cloud products.

RED HAT OPENSTACK PLATFORM

Red Hat OpenStack Platform is a collection of integrated components that provide the functionality to create, deploy, scale, secure, and manage a public or private cloud. Red Hat OpenStack Platform incorporates the capabilities of Red Hat Enterprise Linux with OpenStack to deliver a scalable and secure foundation for building and managing clouds.

Use case:

- A small company or organization with a tightly guarded budget can leverage Red Hat OpenStack Platform to reduce the cost of maintaining the IT infrastructure and at the same time provide a simple and efficient way to access IT services and resources.

Deploying Red Hat OpenStack Platform as a solution would allow for reducing the amounts of physical hardware needed to support their applications and services. As seasonal business peaks, more servers can be deployed using a simple web interface. When business slows, the same web interface can be used to decommission the servers.

The IT budget can be stretched significantly by using virtual servers in Red Hat OpenStack Platform, effectively eliminating the rotating replacement costs for most of the physical servers and workstations. As cost savings are realized and companies or organizations grow, expanding or investing in new cloud technologies may become a priority. Leveraging Red Hat OpenStack Platform to deploy a Platform-as-a-Service (PaaS) solution such as Red Hat OpenShift Container Platform will provide a more flexible way for companies and organizations to offer applications and services to customers both internally and externally.

RED HAT CLOUDFORMS

Red Hat CloudForms manages a variety of cloud technologies, such as Azure, Red Hat OpenStack, Red Hat Virtualization, and VMware. CloudForms alleviates the burden of managing public, private, and hybrid clouds through individual management interfaces, giving cloud administrators a "single pane of glass" view. It also provides an automation engine allowing custom orchestration tasks across the various cloud technologies.

Use case:

- Many organizations have complex and fragmented IT services which require excessive manual processes for tasks such as administration and provisioning. When IT services are fragmented, there are typically separate infrastructures within the organization for basic common services such as mail servers, document processing and storage, research on high-performance computing systems, and management of user accounts.

Centralizing administration and provisioning tasks will eliminate the complexity and fragmentation of IT services. Red Hat CloudForms integrates all of these components into a single administrative portal that reduces manual processing and adds standardization to work environments across the organization.

RED HAT VIRTUALIZATION

Red Hat Virtualization (RHV) is a virtualization platform that allows centralized management of hosts, virtual servers, and desktops across an enterprise data center featuring live migration, high availability, system scheduling, power management, image management, snapshots, thin provisioning, and monitoring.

Use cases:

- Many organizations use Red Hat Virtualization to reduce physical server sprawl by consolidating those servers as virtual machines.
Applications running on obsolete and unsupported hardware are also good candidates for such consolidation.
- Development and test environments often require resources that must be delivered quickly. Those resources usually have a short lifespan; the time for a test to run for example. The Red Hat Virtualization web user portal can allow developers to directly manage the life cycle of their environment, from system provisioning to retirement.

RED HAT OPENSHIFT CONTAINER PLATFORM

Red Hat OpenShift Container Platform, is a Platform-as-a-Service (PaaS) that provides developers and IT organizations with a cloud application platform for deploying new applications on secure, scalable resources with minimal configuration and management overhead.

Built on Red Hat Enterprise Linux and Kubernetes, Red Hat OpenShift Container Platform provides a secure and scalable multiple project operating system for today's enterprise-class applications, while providing integrated application runtimes and libraries. Red Hat OpenShift Container Platform brings a robust, flexible and scalable PaaS platform to customer data centers, enabling organizations to implement a private PaaS that meets security, privacy, compliance, and governance requirements.

Deploying a foundation using Red Hat OpenStack Platform and Red Hat OpenShift Container Platform provides organizations with advancement and growth opportunities.

Use case:

- A fictional company that creates applications for communication systems requires an environment where it can develop and test its product line. Red Hat OpenShift Container Platform can provide developers with a way to quickly choose the tools needed for developing applications. Developers will have immediate access to development environments and hardware conflicts are eliminated when the applications transition to testing by using containers to identically mirror the development environments. This process provides the company with faster go-to-market times for their products.

In addition to improved developer productivity through building and deploying applications faster, the company has the ability to auto-scale Red Hat OpenShift Container Platform and leverage integration with services such as identity management, networking, and storage.

RED HAT SATELLITE

Red Hat Satellite provides the necessary infrastructure and functionality for successful large-scale management of Red Hat Enterprise Linux systems.

Use case:

- Companies and organizations with deployments of Red Hat Enterprise Linux ranging from just a few to many instances can benefit from using Red Hat Satellite. Its systems management tool can be used to configure new systems and provide software updates from Red Hat. It

serves as a local repository of software content and a central point of management for Red Hat entitlements. Red Hat Satellite also performs provisioning and configuration management of systems to adhere to predefined standard operating environments.

With the addition of the Satellite capsule server to its architecture, Red Hat Satellite can scale effectively to meet the demands of large enterprises. With the correct design, Satellite Server, in conjunction with Satellite capsule servers, delivers solid performance in the face of increasing workloads, even across a geographically distributed environment.

Companies and organizations further benefit from the ability of Red Hat Satellite to deploy complex cloud systems such as Red Hat OpenStack Platform, Red Hat OpenShift Container Platform, and Red Hat Virtualization.

RED HAT ENTERPRISE LINUX

Red Hat Enterprise Linux provides core operating system functions and capabilities for application infrastructures.

Use case:

- The requirements for an operating system remain the same, whether it is being used by an individual or by a large organization. Common expectations for an operating systems include features such as being fast, reliable, secure, flexible, and scalable. Additional requirements may be to support complex application development, cloud infrastructures such as virtualization and container deployments, complex networking, and storage solutions.

Red Hat Enterprise Linux is designed to handle any infrastructure choices necessary to meet both the basic needs of an individual user and the complex requirements of a large organization. Between individuals or organizations, needs range from simple to complex, starting with a stable operating system that can support goals such as developing network intensive applications, having scalable data repositories, and ensuring high levels of security (SELinux) and reliability.

Additionally, Red Hat Enterprise Linux provides a powerful and scalable foundation for deploying other Red Hat products including cloud solutions such as Red Hat OpenStack Platform Red Hat OpenShift Container Platform, and Red Hat Virtualization.

RED HAT CLOUD INFRASTRUCTURE

Red Hat Cloud Infrastructure integrates multiple Red Hat products optimized to work together. This product solution provides virtualization, private cloud, public cloud interoperability, storage, and a common management framework.

Components

- Red Hat Enterprise Linux
- Red Hat OpenStack Platform
- Red Hat Ceph Storage
- Red Hat Satellite
- Red Hat Virtualization
- Red Hat CloudForms
- Red Hat Insights

Red Hat Cloud Infrastructure makes it possible for companies and organizations to build, maintain, and scale their own open private hybrid cloud.

- *Open* means that Red Hat Cloud Infrastructure can make use of new open source technologies as they become available.
- *Private* means that Red Hat Cloud Infrastructure makes it possible for you to secure your critical data within infrastructure that you own and control.

- *Hybrid* means that Red Hat Cloud Infrastructure integrates with cloud technologies that you are already using, or that you might use in the future. These technologies include Red Hat Virtualization and Red Hat OpenStack Platform. It also means that you can direct a portion of your workload to public cloud providers, such as Amazon EC2, Microsoft Azure.

RED HAT CLOUD SUITE

Red Hat Cloud Suite integrates multiple Red Hat products optimized to work together. This product provides virtualization, private cloud, public cloud interoperability, container-based application development, storage, and a common management framework.

Components

- Red Hat Enterprise Linux
- Red Hat OpenStack Platform
- Red Hat Ceph Storage
- Red Hat Satellite
- Red Hat Virtualization
- Red Hat OpenShift Container Platform
- Red Hat CloudForms

Red Hat Cloud Suite includes all the components available in Red Hat Cloud Infrastructure, as well as Red Hat OpenShift Container Platform, the Platform-as-a-Service (PaaS) solution.



REFERENCES

Further information is available in the Introduction section of the *Product Guide* for Red Hat Cloud Infrastructure at
<https://access.redhat.com/documentation/en/red-hat-cloud-infrastructure/>

The Introduction section of the *Product Guide* for Red Hat Cloud Suite at
<https://access.redhat.com/documentation/en/red-hat-cloud-suite/>

Multiple sections of the *Product Documentation* for Red Hat OpenStack Platform at
<https://access.redhat.com/documentation/en/red-hat-openstack-platform/>

Multiple sections of the *Product Documentation* for Red Hat CloudForms at
<https://access.redhat.com/documentation/en/red-hat-cloudforms/>

Multiple sections of the *Product Documentation* for Red Hat Virtualization at
<https://access.redhat.com/documentation/en/red-hat-virtualization/>

Multiple sections of the *Product Documentation* for Red Hat OpenShift Container Platform at
<https://access.redhat.com/documentation/en/openshift-container-platform/>

Multiple sections of the *Product Documentation* for Red Hat Satellite at
<https://access.redhat.com/documentation/en/red-hat-satellite/>

Multiple sections of the *Product Documentation* for Red Hat Enterprise Linux at
<https://access.redhat.com/documentation/en/red-hat-enterprise-linux/>

ILLUSTRATING RED HAT CLOUD PRODUCT USE CASES

Choose the correct answers to the following questions:

- ▶ **1. Which Red Hat product uses a capsule server to scale to the demands of large enterprises?**
 - a. Red Hat OpenShift Container Platform
 - b. Red Hat Virtualization
 - c. Red Hat Enterprise Linux
 - d. Red Hat CloudForms
 - e. Red Hat Satellite

- ▶ **2. Which Red Hat product provides the functionality that allows management for large-scale deployments of Red Hat Enterprise Linux servers and workstations?**
 - a. Red Hat Satellite
 - b. Red Hat OpenStack Platform
 - c. Red Hat Enterprise Linux
 - d. Red Hat OpenShift Container Platform

- ▶ **3. Which two Red Hat products provide the functionality to eliminate the necessity of managing multiple clouds through individual management interfaces? (Choose two.)**
 - a. Red Hat OpenShift Container Platform
 - b. Red Hat Cloud Infrastructure
 - c. Red Hat Enterprise Linux
 - d. Red Hat CloudForms
 - e. Red Hat Satellite

- ▶ **4. Which four Red Hat products are included in the Red Hat Cloud Infrastructure offering from Red Hat? (Choose four.)**
 - a. Red Hat OpenStack Platform
 - b. Red Hat OpenShift Container Platform
 - c. Red Hat Virtualization
 - d. Red Hat Enterprise Linux
 - e. Red Hat CloudForms
 - f. Red Hat Satellite

ILLUSTRATING RED HAT CLOUD PRODUCT USE CASES

Choose the correct answers to the following questions:

- ▶ **1. Which Red Hat product uses a capsule server to scale to the demands of large enterprises?**
 - a. Red Hat OpenShift Container Platform
 - b. Red Hat Virtualization
 - c. Red Hat Enterprise Linux
 - d. Red Hat CloudForms
 - e. Red Hat Satellite

- ▶ **2. Which Red Hat product provides the functionality that allows management for large-scale deployments of Red Hat Enterprise Linux servers and workstations?**
 - a. Red Hat Satellite
 - b. Red Hat OpenStack Platform
 - c. Red Hat Enterprise Linux
 - d. Red Hat OpenShift Container Platform

- ▶ **3. Which two Red Hat products provide the functionality to eliminate the necessity of managing multiple clouds through individual management interfaces? (Choose two.)**
 - a. Red Hat OpenShift Container Platform
 - b. Red Hat Cloud Infrastructure
 - c. Red Hat Enterprise Linux
 - d. Red Hat CloudForms
 - e. Red Hat Satellite

- ▶ **4. Which four Red Hat products are included in the Red Hat Cloud Infrastructure offering from Red Hat? (Choose four.)**
 - a. Red Hat OpenStack Platform
 - b. Red Hat OpenShift Container Platform
 - c. Red Hat Virtualization
 - d. Red Hat Enterprise Linux
 - e. Red Hat CloudForms
 - f. Red Hat Satellite

VERIFYING OPENSTACK SERVICES

OBJECTIVES

After completing this section, students should be able to verify OpenStack services.

RED HAT OPENSTACK PLATFORM SERVICES

Red Hat OpenStack Platform director deploys Red Hat OpenStack Platform using multiple services that together form a Infrastructure-as-a-Service (IaaS) cloud. Each service is composed of several components that can be deployed as containers on a single server or on separate servers depending on available resources and the intended purpose.

The cloud environment can be managed from a web-based interface or using command-line clients. Both methods use an extended API which is available to cloud administrators and cloud users.

Core OpenStack Services

The following table describes the core OpenStack services installed during a typical Red Hat OpenStack Platform director installation.

SERVICE (CODE NAME)	DESCRIPTION
dashboard (<i>horizon</i>)	A web-based interface for managing OpenStack services. It provides a graphical user interface for operations such as launching instances, managing networking, and setting access controls.
Identity service (<i>keystone</i>)	A centralized identity service that provides authentication and authorization for other services. The identity service also provides a central catalog of services running in a particular OpenStack cloud. It supports multiple forms of authentication, including username and password credentials, token-based systems, and Amazon Web Services (AWS) logins. The identity service acts as a single sign-on (SSO) authentication service for users and components.
OpenStack Networking service (<i>neutron</i>)	A service that handles the creation and management of a virtual networking infrastructure in the OpenStack cloud. Elements include networks, subnets, and routers. Advanced services such as firewalls or virtual private networks (VPN) can also be used. Due to OpenStack networking's extensible architecture, users can create their own networks, control traffic, and connect servers to other networks. Various networking technologies are supported.
Block Storage service (<i>cinder</i>)	A service that manages storage volumes for virtual machines. This is persistent block storage for the instances running in the compute service. Snapshots can be taken for backing up data, either for restoring data or to be used to create new block storage volumes. This is often used in instances for storage, such as database files.

Service (Code Name)	Description
Compute service (<i>nova</i>)	A service that manages networks of virtual machines running on nodes, providing virtual machines on demand. The compute service is a distributed component and interacts with the identity service for authentication, the image service for template images, and the dashboard for a graphical OpenStack interface. Compute is designed to scale horizontally on standard hardware, downloading images to launch instances as required. The compute service uses libvirt , qemu , and kvm for the hypervisor.
Image service (<i>glance</i>)	A service that acts as a registry for virtual machine images, allowing users to copy server images for immediate storage. These images can be used as templates when setting up new instances.
Object Store (<i>swift</i>)	A service providing object storage that allows users to store and retrieve files. The object store architecture is distributed to allow for horizontal scaling and to provide redundancy as failure-proofing. Data replication is managed by software, allowing greater scalability and redundancy than dedicated hardware.
Telemetry service (<i>ceilometer</i>)	A centralized source for metering and monitoring data. The multiple components of the telemetry component set provide the capability to meter and bill OpenStack users.
Orchestration service (<i>heat</i>)	A service to orchestrate multiple composite cloud applications using the Amazon Web Services (AWS) CloudFormation template format, through both a Representational State Transfer (REST) API and a CloudFormation-compatible Query API. The software integrates other core components of OpenStack into a one file template system. Templates allow creation of most OpenStack resource types, such as instances, floating IPs, volumes, security groups, and users. Templates also provide advanced functionality such as instance high availability, instance autoscaling, and nested stacks.

The previous table also gives the code name of each service. Developers, documentations, and upstream projects often refer to the services by their code names. Objects on the systems, such as configuration files, log files, and container names, also use these code names. For example, the main log file for the identity service (keystone) is **/var/log/containers/keystone/keystone.log**.

LISTING, VERIFYING, AND MANAGING OPENSTACK SERVICES

Gathering accurate information quickly for OpenStack services is key to maintaining a stable and productive environment.

OpenStack implements each of its service as a collection of containers running on the servers. OpenStack uses **Docker** as its container implementation, and administrators use the **docker** command to manage and monitor the running containers.



NOTE

Red Hat OpenStack Platform still provides the **openstack-status** and the **openstack-service** commands in an OpenStack developer tools repository but no longer provides support for these unmaintained commands. Now that virtually all OpenStack services are containerized, services must be managed with container system tools instead.

List Status Reports for the Running OpenStack Services

The **docker ps** command lists the running containers on the local node.

```
[root@controller ~]# docker ps
CONTAINER ID  IMAGE
COMMAND      CREATED        STATUS          PORTS
NAMES
26b1b4a4d469  172.25.249.200:8787/rhosp13/openstack-cinder-volume:pcmklatest
    "/bin/bash /usr/lo..."  2 days ago   Up About an hour
openstack-cinder-volume-docker-0
fd3839703681  172.25.249.200:8787/rhosp13/openstack-haproxy:pcmklatest
    "/bin/bash /usr/lo..."  2 days ago   Up About an hour
haproxy-bundle-docker-0
89a92625e50f  172.25.249.200:8787/rhosp13/openstack-redis:pcmklatest
    "/bin/bash /usr/lo..."  2 days ago   Up About an hour
redis-bundle-docker-0
de6a285e75a5  172.25.249.200:8787/rhosp13/openstack-mariadb:pcmklatest
    "/bin/bash /usr/lo..."  2 days ago   Up About an hour
galera-bundle-docker-0
31045add88fd  172.25.249.200:8787/rhosp13/openstack-rabbitmq:pcmklatest
    "/bin/bash /usr/lo..."  2 days ago   Up About an hour
rabbitmq-bundle-docker-0
5a4d50ccbf0  172.25.249.200:8787/rhceph/rhceph-3-rhel7:latest
    "/entrypoint.sh"       2 days ago   Up About an hour
ceph-mgr-controller0
7e731d900926  172.25.249.200:8787/rhceph/rhceph-3-rhel7:latest
    "/entrypoint.sh"       2 days ago   Up About an hour
ceph-mon-controller0
ccdf9a3411a7d  172.25.249.200:8787/rhosp13/openstack-neutron-openvswitch-
agent:latest  "kolla_start"           2 days ago   Up About an hour (healthy)
    neutron_ovs_agent
059155019dff  172.25.249.200:8787/rhosp13/openstack-neutron-13-agent:latest
    "kolla_start"           2 days ago   Up About an hour (healthy)
    neutron_13_agent
8cdf54fecbc8  172.25.249.200:8787/rhosp13/openstack-neutron-metadata-agent:latest
    "kolla_start"           2 days ago   Up About an hour (healthy)
    neutron_metadata_agent
33828ea2101d  172.25.249.200:8787/rhosp13/openstack-neutron-dhcp-agent:latest
    "kolla_start"           2 days ago   Up About an hour (healthy)
    neutron_dhcp
16283397e46d  172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "kolla_start"           2 days ago   Up 59 minutes
    nova_metadata
974a110d5fb  172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "kolla_start"           2 days ago   Up About an hour (healthy)
    nova_api
```

...output omitted...

Notice that the **docker** command is only available to the **root** user.

For better readability, you can select the columns to display with the **--format** option and use the **grep** command to limit the output to a specific service. The following example only displays the container names for Neutron.

```
[root@controller ~]# docker ps --format="{{.Names}}" | grep neutron
neutron_ovs_agent
neutron_l3_agent
neutron_metadata_agent
neutron_dhcp
neutron_api
```

Typically, an OpenStack service has several component containers associated with it. The previous output shows that the Networking Service has an API container, **neutron_api**, to manage the API requests, plus additional components such as a DHCP agent, a metadata agent, and an Open vSwitch agent (**ovs**).

These internal components communicate between them through a message queue provided by the *RabbitMQ* software. Additionally, a *Galera MySQL* database server persistently stores the data for the OpenStack services. OpenStack also uses containers for all these support services.

```
[root@controller ~]# docker ps --format="{{.Names}}" | grep -e rabbitmq -e galera
galera-bundle-docker-0
rabbitmq-bundle-docker-0
```

Verifying the OpenStack Services

The **docker ps** command only displays the running containers. To list the failed and stopped containers, you can use the **--all** (or **-a**) option.

```
[root@controller ~]# docker ps -a --format="{{.Names}}\t{{.Status}}"
openstack-cinder-volume-docker-0    Up 2 hours
haproxy-bundle-docker-0            Up 2 hours
redis-bundle-docker-0              Up 2 hours
galera-bundle-docker-0            Up 2 hours
rabbitmq-bundle-docker-0          Up 2 hours
ceph-mgr-controller0              Up 2 hours
ceph-mon-controller0              Up 2 hours
nova_api_discover_hosts-di5gl659 Exited (0) 17 hours ago
cinder_volume_init_bundle-qpe85l2j Exited (0) 17 hours ago
nova_api_discover_hosts           Exited (0) 2 days ago
cinder_volume_init_bundle         Exited (0) 2 days ago
neutron_ovs_agent                 Up 2 hours (healthy)
neutron_l3_agent                  Up 2 hours (healthy)
...output omitted...
```

The **Exited** containers, with a **(0)** return code, are not in error. They are *Init* containers that are executed once at system boot and then exist. They initialize and prepare the system for the other containers implementing the OpenStack services.

Docker monitors the running containers and takes care of restarting them when they fail.

Managing the OpenStack Services

Administrators can stop, start, and restart containers with the `docker stop container_name`, `docker start container_name`, and `docker restart container_name` commands. For instance, when you modify the configuration file of an OpenStack service, you need to restart the associated containers.

Managing the OpenStack Services

The following steps outline the process you can follow to modify the configuration of the `nova_api` container:

1. Display the status of the `neutron_ovs_agent` container with the `docker ps | grep nova_api` command on both `controller0` and `compute0` nodes.
2. Display the log files for keystone using the `docker logs keystone` command.
3. Display the status of the `nova_api` container with the `docker ps | grep nova_api` command.
4. Edit the compute service configuration file, `/var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf`.
5. Restart the `nova_api` container with the `docker restart nova_api` command.
6. Use the `docker ps | grep nova_api` command to verify that the container has restarted. The `STATUS` column indicates that the container is up.



REFERENCES

`docker-ps(1)` man page.

Further information is available in the Software section of the *Product Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/product_guide/ch-rhosp-software

VERIFYING OPENSTACK SERVICES

In this exercise, you will explore the use of commands to view status information and manage services used in Red Hat OpenStack Platform.

OUTCOMES

You should be able to:

- Show status overview of installed OpenStack services.
- View and control enabled OpenStack services.

Log in to **workstation** as **student** using **student** as the password. From **workstation**, run **lab verify-osp-services setup**, which ensures OpenStack services are running.

```
[student@workstation ~]$ lab verify-osp-services setup
```

- ▶ 1. From **workstation**, open a terminal and use SSH to access **controller0**.

```
[student@workstation ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$
```

- ▶ 2. Using the **docker ps** command list the running containers on the **controller0** node.

2.1. The **docker ps** command requires root privileges. Use **sudo -i** to become root.

```
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

2.2. Use the **docker ps** command to list the running services on the **controller0** node.

```
[root@controller0 ~]# docker ps
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS
NAMES
...output omitted...
8bddfa6bad7e      172.25.249.200:8787/rhosp13-beta/openstack-neutron-dhcp-
agent:latest       "kolla_start"          6 days ago         Up 22 hours
(healthy)           neutron_dhcp
ffa52e3683b       172.25.249.200:8787/rhosp13-beta/openstack-nova-api:latest
                  "kolla_start"          6 days ago         Up 22 hours
                  nova_metadata
15dabb3a0e80       172.25.249.200:8787/rhosp13-beta/openstack-nova-api:latest
                  "kolla_start"          6 days ago         Up 22 hours (healthy)
                  nova_api
```

```
ce408b6da970      172.25.249.200:8787/rhosp13-beta/openstack-glance-api:latest
                  "kolla_start"          6 days ago        Up 22 hours (healthy)
                      glance_api
...output omitted...
```

- 2.3. As you can see the output is difficult to read. For better readability, select the columns to display using the **--format** option with the **grep** command to limit the output.

```
[root@controller0 ~]# docker ps --format="{{.Names}}" | grep swift
swift_proxy
swift_container_auditor
swift_object_expirer
swift_object_updater
swift_container_replicator
swift_account_auditor
swift_account_server
swift_object_replicator
swift_container_server
swift_rsync
swift_account_reaper
swift_account_replicator
swift_object_auditor
swift_object_server
swift_container_updater
```

- 2.4. There are a few other options to make the output of the **docker ps** command more readable. For example, using the **json** format and piping the output to **jq**. Or alternatively, defining a custom format using the **GO** language.

```
[root@controller0 ~]# docker ps --format '{{ json .}}' | jq .
...output omitted...
{
  "Status": "Up 22 hours",
  "Size": "0 B",
  "RunningFor": "6 days",
  "Ports": "",
  "Networks": "host",
  "Command": "\"kolla_start\"",
  "CreatedAt": "2018-06-01 12:16:26 +0000 UTC",
  "ID": "897056bbee32",
  "Image": "172.25.249.200:8787/rhosp13-beta/openstack-horizon:latest",
  "Labels": "summary=Red Hat OpenStack Platform 13.0 horizon,vendor=Red Hat,...",
  ...
  "Names": "horizon"
}
...output omitted...
[root@controller0 ~]# docker ps --format \
'CONTAINER ID: {{.ID}}
IMAGE: {{.Image}}
COMMAND: {{.Command}}
CREATED: {{.CreatedAt}}
STATUS: {{.Status}}
NAMES: {{.Names}}
'

CONTAINER ID: 0ede0883fdf1
```

```
IMAGE: 172.25.249.200:8787/rhosp13-beta/openstack-nova-scheduler:latest
COMMAND: "kolla_start"
CREATED: 2018-06-01 12:23:40 +0000 UTC
STATUS: Up 22 hours (healthy)
NAMES: nova_scheduler
...output omitted...
```

- 3. Use the **docker ps** command to verify the status of the OpenStack services.

```
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}"
ceph-mgr-controller0      Up 22 hours
nova_api_discover_hosts   Exited (0) 6 days ago
cinder_volume_init_bundle  Exited (0) 6 days ago
neutron_ovs_agent          Up 22 hours (healthy)
neutron_l3_agent           Up 22 hours (healthy)
neutron_metadata_agent     Up 22 hours (healthy)
neutron_dhcp               Up 22 hours (healthy)
nova_metadata              Up 22 hours
nova_api                   Up 22 hours (healthy)
glance_api                 Up 22 hours (healthy)
swift_proxy                Up 22 hours (healthy)
...output omitted...
```



NOTE

Containers with a status of **Exited** are not in error. They are *Init* containers that are executed at system boot and then exist.

- 4. Using the **docker** command with the **stop** and **start** options, manage the **swift_proxy** service.

```
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" \
| grep swift_proxy
swift_proxy    Up 23 hours (healthy)
[root@controller0 ~]# docker stop swift_proxy
swift_proxy
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" \
| grep swift_proxy
swift_proxy    Exited (0) 53 seconds ago
[root@controller0 ~]# docker start swift_proxy
swift_proxy
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" \
| grep swift_proxy
swift_proxy    Up 22 seconds (health: starting)
```

- 5. Exit from the **controller0** virtual machine.

```
[root@controller0 ~]# exit
logout
[heat-admin@controller0 ~]$ exit
logout
[student@workstation ~]$
```

Cleanup

From **workstation**, run the **lab verify-osp-services cleanup** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab verify-osp-services cleanup
```

This concludes the guided exercise.

DESCRIBING CLOUD COMPUTING

Choose the correct answers to the following questions:

► 1. Which two characteristics are related to traditional computing workloads? (Choose two.)

- a. Client-server architecture
- b. Failover is built into an application
- c. Single machines handle the workload
- d. To scale up requires adding more machines
- e. Scaling is built into an application

► 2. In Red Hat OpenStack Platform which user is associated with the admin role in the admin project by default?

- a. manager
- b. superuser
- c. administrator
- d. Cloud admin
- e. admin

► 3. Which three characteristics are true about system virtualization? (Choose three.)

- a. Virtual machines are isolated from one another
- b. From the virtual machine operating system's perspective, it is running on its own private hardware
- c. From the perspective of a single hypervisor, all of its virtual machines run the same private operating system
- d. Several virtual machines can be deployed on a single computer

► 4. Which three Red Hat products provide administrators with a single-pane-view for managing public, private, and hybrid clouds? (Choose three.)

- a. Red Hat Cloud Infrastructure
- b. Red Hat Cloud Suite
- c. Red Hat Satellite
- d. Red Hat CloudForms
- e. Red Hat OpenStack Platform

- 5. Which command allows administrators to list the running containers for the Compute service?
- a. `systemctl status openstack-compute`
 - b. `systemctl status nova`
 - c. `openstack-status`
 - d. `docker ps | grep nova`
 - e. `docker ps nova`

DESCRIBING CLOUD COMPUTING

Choose the correct answers to the following questions:

► 1. Which two characteristics are related to traditional computing workloads? (Choose two.)

- a. Client-server architecture
- b. Failover is built into an application
- c. Single machines handle the workload
- d. To scale up requires adding more machines
- e. Scaling is built into an application

► 2. In Red Hat OpenStack Platform which user is associated with the admin role in the admin project by default?

- a. manager
- b. superuser
- c. administrator
- d. Cloud admin
- e. admin

► 3. Which three characteristics are true about system virtualization? (Choose three.)

- a. Virtual machines are isolated from one another
- b. From the virtual machine operating system's perspective, it is running on its own private hardware
- c. From the perspective of a single hypervisor, all of its virtual machines run the same private operating system
- d. Several virtual machines can be deployed on a single computer

► 4. Which three Red Hat products provide administrators with a single-pane-view for managing public, private, and hybrid clouds? (Choose three.)

- a. Red Hat Cloud Infrastructure
- b. Red Hat Cloud Suite
- c. Red Hat Satellite
- d. Red Hat CloudForms
- e. Red Hat OpenStack Platform

- 5. Which command allows administrators to list the running containers for the Compute service?
- a. `systemctl status openstack-compute`
 - b. `systemctl status nova`
 - c. `openstack-status`
 - d. `docker ps | grep nova`
 - e. `docker ps nova`

SUMMARY

In this chapter, you learned:

- Cloud computing leverages virtualization to provide on-demand network access to a shared pool of configurable computing resources.
- Virtual machines and containers, although different, are both integral parts of cloud computing solutions. Red Hat Virtualization provides a platform that allows centralized management of virtual machine servers and desktops across an enterprise data center. Red Hat OpenShift Container Platform provides developers with a cloud application platform for deploying scalable, fast-to-market applications through the use of containers.
- Red Hat products such as Red Hat Enterprise Linux, Red Hat Virtualization, Red Hat Satellite, Red Hat OpenStack Platform, Red Hat OpenShift Container Platform, and Red Hat CloudForms can all function as individual product solutions . However, Red Hat products become even more powerful when combined together to provide solutions in large-scale enterprise environments using products such as Red Hat Cloud Infrastructure and Red Hat Cloud Suite.
- Gathering accurate information quickly is key to maintaining a stable productive cloud computing environment using tools, such as the **docker ps** command.

CHAPTER 4

MANAGING LINUX NETWORKS

GOAL

Manage Linux networks and bridges.

OBJECTIVES

- Describing networking concepts.
- Manage Linux network interfaces.
- Implement Linux bridges.
- Implement Open vSwitch (OVS) bridges.

SECTIONS

- Describing Networking Concepts (and Quiz)
- Managing Network Interfaces (and Guided Exercise)
- Implementing Linux Bridges (and Guided Exercise)
- Implementing Open vSwitch Bridges (and Guided Exercise)

LAB

- Managing Linux Networks

DESCRIBING NETWORKING CONCEPTS

OBJECTIVES

After completing this section, students should be able to:

- Explain networking concepts such as IPv4 and MTU.
- Discuss *Software-Defined Networking (SDN)*.
- Discuss networking technologies such as bridges and tunnels.

IPV4 NETWORKING

The TCP/IP standards follow a four-layer network model specified in RFC1122:

Application Layer

Each application has specifications for communication so that clients and servers can communicate across platforms. Common protocols include SSH, which provides remote login to the servers; HTTPS, which provides secure web access; NFS or CIFS, which provide file sharing; and SMTP, which provides electronic mail delivery services.

Transport Layer

TCP and UDP are the two transport layer protocols. TCP is a reliable connection-oriented communication protocol, whereas UDP is a connectionless *datagram* protocol. Applications communicate by exchanging either TCP or UDP packets. A list of well-known and registered ports can be found in the **/etc/services** file.

Internet Layer

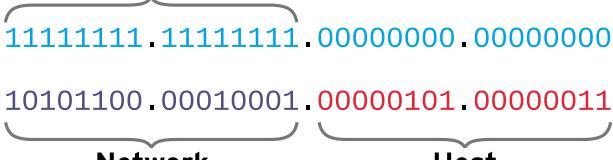
The internet, or network layer, carries data from the source host to the destination host. Each host has an IP address and a prefix used to determine the network addresses. Routers, which are used to connect multiple networks, ensure the routing of the traffic between such networks.

ICMP is a control protocol that belongs to this layer. Instead of using ports, this protocol uses packet *types*. The **ping** utility is an example of a command that sends ICMP packets to test the connectivity between two systems.

Link Layer

The link layer, or media access layer, provides the connection to physical media. The most common types of networks are wired Ethernet (IEEE standard 802.3) and wireless WLAN (IEEE standard 802.11). Each physical device has a hardware address, or *MAC* address, which is used to identify the destination of packets on the local network segment.

IP Address:
172.17.5.3 = 10101100.00010001.00000101.00000011

Netmask:
255.255.0.0 = 

IP Address:
192.168.5.3 = 11000000.10101000.00000101.00000011

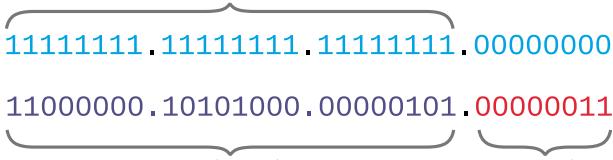
Netmask:
255.255.255.0 = 

Figure 4.1: IPv4 addressing and netmasks

IPv4 Addresses

An IPv4 address is a 32-bit number expressed in decimal as four *octets*, ranging in value from 0 to 255, separated by dots. The address is divided into two parts: the network part and the host part. All hosts on the same subnet can communicate with each other directly without a router, and have the same network part. However, two hosts on the same subnet cannot have the same host part, because the host part of the address identifies the different hosts on a subnet.

In the modern internet, the size of an IPv4 subnet is variable. To know which part of an IPv4 address is the network part and which is the host part, administrators must know the *netmask* that is assigned to the subnet. The netmask indicates how many bits of the IPv4 address belong to the subnet. The more bits that are available for the host part, the more hosts can be on the subnet.

The lowest possible address on a subnet is sometimes referred as the *network address*. The highest possible address on a subnet is used for broadcast messages in IPv4, and is called the *broadcast address*. Network masks are expressed in two forms. The older syntax for a netmask that uses 24 bits for the network part would read 255.255.255.0. The newer syntax, called Classless inter-domain routing (CIDR) notation, would specify a *network prefix* of /24. Both forms convey the same information; the number of leading bits in the IP address that contribute to its network address.

SOFTWARE-DEFINED NETWORKING

Software-defined networking (SDN) is a networking model that allows network administrators to manage network services through the abstraction of several networking layers. SDN decouples the software that handles the traffic, called the *control plane*, and the underlying mechanisms that route the traffic, called the *data plane*. SDN enables communication between the control plane and the data plane. For example, the OpenFlow project, combined with the OpenDaylight project, provides such implementation.

SDN does not change the underlying protocols used in networking; rather, it enables the utilization of application knowledge to provision networks. Networking protocols, such as TCP/IP and Ethernet standards, rely on manual configuration by administrators for applications. They do not manage networking applications, such as their network usage, the end point requirements, or how much and how fast the data needs to be transferred. The goal of SDN is to extract knowledge of how an application is being used by the application administrator or the application's configuration data itself.

Ethernet Headers

Each network packet possesses a header, which has a length of 14 octets, comprised of the source MAC address from which the packet originates, the destination MAC address, and the type of the Ethernet frame, such as Ethernet II, or IEEE 802.2 LLC. Optionally, the Ethernet frame may encode an **IEEE 802.1Q** tag, which defines the identifier for a *Virtual Local Area Network (VLAN)*. An Ethernet domain is a layer 2 network (layer two of seven defined by the OSI model of network computing), or simply a network. Traditionally, MAC addresses were assigned by hardware manufacturers. Each manufacturer assigned a unique set of bits for a prefix, and generated the rest of the bits for each network interface.

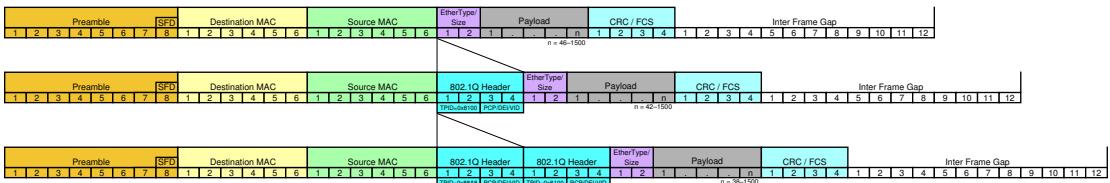


Figure 4.2: Ethernet headers in a network packet



IMPORTANT

Although MAC uniqueness is expected, there are cases where it does not happen. Hardware vendors might reuse MAC addresses. If a MAC address is set by an administrator, or if a service such as OpenStack networking is configured to autogenerated MAC addresses, overlapping MAC addresses are possible. Using multiple devices with the same MAC address on the same Ethernet domain will prevent the packets from being correctly routed.

Bridges

A network bridge is a network device that connects multiple network segments. Bridges can connect multiple devices, and each device can send Ethernet frames to other devices without having the frame removed and replaced by a router. Bridges keep the traffic isolated, and in most cases, the switch is aware of which MAC addresses are accessible via which ports. Switches monitor network activity and maintain a MAC *learning table*.

Broadcast Domains

A broadcast domain is a logical division of a computer network, in which all nodes can reach each other by broadcast at the data link layer. A broadcast domain can be within the same LAN segment or it can be bridged to other LAN segments. Any device that belongs to a network segment can send a broadcast frame that will be repeated to all other devices in the segment.

Integration of IP and Ethernet

Users interact with network services through a high level of abstraction. For example, they only know the URL of a web site that they want to access, which is the domain name for the website, such as **redhat . com**. The web browser interacts with the underlying network implementation, which reaches a DNS server in order to find the IP address that matches the domain name. Even though the IP address is found, the domain name resolution does not return the MAC address. If the server is on the same local network as the user's computer, the system sends Ethernet broadcast packets to determine if any of the systems are configured with the IP address that the user want to connect to. The corresponding system responds with an Ethernet frame that includes the MAC address.

The **Address Resolution Protocol (ARP)** is a telecommunication protocol used for resolution of network layer addresses to link layer addresses. The ARP protocol converts a network address to a physical address, such as an Ethernet address.



NOTE

Network switches can store ARP requests and ARP responses in their MAC learning table. If another ARP request is sent for the same IP address from a different client, the switch can avoid flooding the broadcast over all its ports by using the data stored about the IP address.

The following describes the handling of network packets:

- The client system does a computation based on its own IP address, its subnet, the remote IP address, and the remote subnet.
- If the networks match between the client and the server, the client concludes that the destination is local. If the networks do not match, the client concludes that the destination is remote.
- If the resource is remote, the system forwards the traffic to the default router using the MAC address of the router and IP address of the destination, as set in the Ethernet frame.
- If the destination is located through a broadcast domain directly attached to the router, it performs the same process as the client, which is interacting with the ARP protocol to forward the frame.

A single IP packet can travel through multiple routers to reach its destination, and each time, the Ethernet frame is removed and regenerated by each router. In such cases, the original IP packet is encapsulated in the Ethernet frame. Many Ethernet broadcast domains can be connected with IP routers that allow access to foreign network types used for networks such as *Wide Area Networks (WANs)*, which do not forward broadcast packets.

VXLAN Tunnels

Virtual eXtensible LAN (VXLAN) is a network virtualization technology which solves the scalability problems associated with large cloud computing deployments. It increases scalability up to 16 million logical networks and allows the adjacency of layer 2 links across IP networks. The VXLAN protocol encapsulates L2 networks and tunnels them over L3 networks.

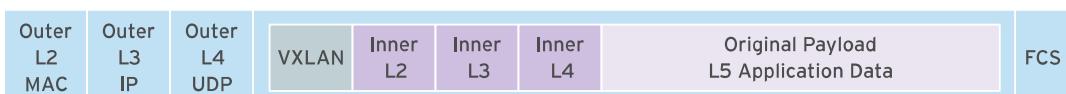


Figure 4.3: VXLAN Ethernet header

Virtual LAN

Administrators partition single layer 2 networks to create multiple broadcast domains that are mutually isolated, so that packets only pass between them via routers. This segregation is known as *Virtual Local Area Networks (VLANs)*. VLANs provide segmentation services traditionally offered by routers in LAN configurations. VLANs address issues such as scalability, security, and network management. Routers in VLAN topologies provide broadcast filtering, security, address summary, and traffic-flow management. VLANs can create multiple layer 3 networks on single physical segments. For example, a DHCP server available to a unsegmented switch will serve any DHCP client host on that switch. By using VLANs, the switch network can be isolated, such that some hosts see the DHCP server and others are isolated in a different segment.

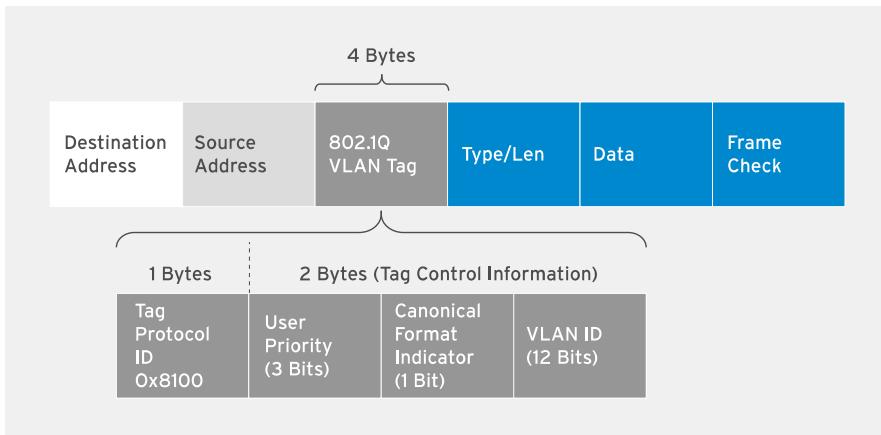


Figure 4.4: VLAN header

In a VLAN environment, VLANs typically map one-to-one with IP subnets, although it is possible to create multiple subnets on a single VLAN. With VLANs, administrators control traffic and react quickly to relocations. VLANs provide the flexibility to adapt to changes in network requirements and to facilitate simplified network management.

Network Processors

Traditionally, routers were similar to software routers. They moved network packets into their memory, performed various lookups, and then moved the data from memory to the outbound network interface. Switches used to be faster than routers, but modern switches and routers use the same hardware. Many hardware vendors provide switches and routers that can be configured on a per-port basis to either perform switching functions or routing functions.

The current generation of hardware uses the concept of a *flow table*. Instead of tracking routes and MAC addresses, modern switches and routers have a single lookup table that can match any criteria, such as the source MAC address, the IP address of the destination MAC address, the IP address, or the TCP port, and then decide to either forward the packet or to drop it. Network capabilities, such as statistics, are maintained in the flow table. Network processors provide the support for the flow table.

While network processors are a great innovation, network protocols have changed little. Instead of developing new protocols for network processors, routers and switches use traditional routing to determine the network flow and inject flow rules into the switch. The first packet in a TCP/IP stream must often be processed by software, later packets are forwarded to destinations using flow rules created in the switch.



REFERENCES

Further information is available in the *Networking Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

Further information is available in the *Network Functions Virtualization Product Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_product_guide/

Further information is available in the *Network Functions Virtualization Planning Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_planning_and_configuration_guide/

DESCRIBING NETWORKING CONCEPTS

Choose the correct answers to the following questions:

- ▶ **1. Which network layer includes the HTTPS protocol, used for OpenStack API communication?**
 - a. Transport Layer
 - b. Internet Layer
 - c. Application Layer
 - d. Link Layer

- ▶ **2. A network bridge device belongs to which layer of the networking model?**
 - a. Transport Layer
 - b. Internet Layer
 - c. Application Layer
 - d. Link Layer

- ▶ **3. Which three statements are true for a network bridge? (Choose three.)**
 - a. A network bridge is a network device that connects multiple network segments.
 - b. A network bridge forwards traffic between networks based on the IP address.
 - c. A network bridge forwards traffic between networks based on MAC addresses.
 - d. A network bridge makes forwarding decisions based on tables of MAC addresses which it builds by learning what hosts are connected to each network.

- ▶ **4. Which network protocol uses a broadcast to find the MAC address for a known IP address?**
 - a. ARP
 - b. SSH
 - c. BOOTP
 - d. ICMP

- ▶ **5. Which technology deploys virtual routers and switches in a virtualized environment?**
 - a. VLAN
 - b. VXLAN
 - c. ARP
 - d. SDN

► 6. Which networking protocol creates multiple networks using 802.1Q tags that correspond with the physical network?

- a. VLAN
- b. VXLAN
- c. ARP
- d. SDN

DESCRIBING NETWORKING CONCEPTS

Choose the correct answers to the following questions:

- ▶ **1. Which network layer includes the HTTPS protocol, used for OpenStack API communication?**
 - a. Transport Layer
 - b. Internet Layer
 - c. Application Layer
 - d. Link Layer

- ▶ **2. A network bridge device belongs to which layer of the networking model?**
 - a. Transport Layer
 - b. Internet Layer
 - c. Application Layer
 - d. Link Layer

- ▶ **3. Which three statements are true for a network bridge? (Choose three.)**
 - a. A network bridge is a network device that connects multiple network segments.
 - b. A network bridge forwards traffic between networks based on the IP address.
 - c. A network bridge forwards traffic between networks based on MAC addresses.
 - d. A network bridge makes forwarding decisions based on tables of MAC addresses which it builds by learning what hosts are connected to each network.

- ▶ **4. Which network protocol uses a broadcast to find the MAC address for a known IP address?**
 - a. ARP
 - b. SSH
 - c. BOOTP
 - d. ICMP

- ▶ **5. Which technology deploys virtual routers and switches in a virtualized environment?**
 - a. VLAN
 - b. VXLAN
 - c. ARP
 - d. SDN

► 6. Which networking protocol creates multiple networks using 802.1Q tags that correspond with the physical network?

- a. VLAN
- b. VXLAN
- c. ARP
- d. SDN

MANAGING NETWORK INTERFACES

OBJECTIVE

After completing this section, students should be able to manage Linux network interfaces.

CONFIGURING NETWORK INTERFACES

Administrators can configure network interfaces to use a persistent configuration. Persistent settings are stored in the configuration files located in `/etc/sysconfig/network-scripts/`. The files in the directory are named `ifcfg-NAME`, where `NAME` refers to the name of the device that the configuration file controls. For example, the interface file named `ifcfg-eth0` defines the network configuration for the `eth0` device. The following table explains some of the variables found in the files.

Network Interface Options

SETTING	PURPOSE
<code>IPADDRn=XXX.XXX.XXX.XXX</code>	Defines the IP address to set for the device. You can specify different values of <code>n</code> to create multiple IP addresses on the same interface.
<code>PREFIXn=XX</code>	Defines the IP netmask, which is a required value for each IP address.
<code>GATEWAY=XXX.XXX.XXX.XXX</code>	Defines an optional gateway for the network, which must be reachable via the network interface.

Other Options

SETTING	PURPOSE
<code>DEVICE=NAME</code>	Specifies the name of the network device.
<code>ONBOOT=yes or no</code>	Controls the interface boot mode. If set to <code>no</code> , the interface will not automatically activate at boot time. To enable the interface, bring it up manually.

The following is a list of additional options which can be set in the network configuration files.

- The `NM_CONTROLLED` option indicates if the network interface is managed by the `NetworkManager` service.
- The `HWADDR` option binds the configuration to the NIC with the MAC address defined by this option.
- The `MACADDR` option specifies the MAC address of the network interface. Using this option overrides the MAC address assigned to the physical network device. Administrators must not use this directive with the `HWADDR` setting.
- The `BOOTPROTO` option defines the protocol to use at boot time, such as `dhcp`, `none`, or `static`.

- The **DNSn** option specifies the IP address of the DNS name server to use for domain name resolution.

The following is an example of a network configuration file, which defines the network configuration for the **eth0** device.

```
# Device Control
DEVICE=eth0
ONBOOT=yes
NM_CONTROLLED=yes
MACADDR=52:54:00:bd:b7:d2

# DHCP IP addressing
BOOTPROTO=dhcp

# Alternative static IP addressing
BOOTPROTO=static
IPADDR=172.25.249.25
PREFIX=24
GATEWAY=172.25.249.1
DNS1=172.25.250.254
```

Network Interface Naming

Red Hat Enterprise Linux 7 is the first release of RHEL which assigns network interface names using systemd's Predictable Network Interface naming scheme. This is described in detail in the RHEL 7 Networking Guide chapter on Consistent Device Naming. Red Hat strongly recommend that the new RHEL7 naming conventions are used.

At boot time, interfaces are *always* assigned ethX style names by the kernel, with the number assigned in the order that devices are detected in this boot event. Since the kernel has no ability to ensure drivers or interfaces are discovered in the same order every time, it is necessary to implement an additional OS function to rename interfaces in a predictable way such that each interface is assigned the same name on subsequent boots. In RHEL 7, this function is provided by systemd's **Predictable Network Interface** feature.

A previous alternate naming scheme, **biosdevname**, was designed and implemented by DELL using unique network interface information presented by the system BIOS. Only DELL hardware consistently provides the necessary information needed for biosdevname to work. Non-DELL systems can enable this scheme by installing the **biosdevname** package and setting the **biosdevname=1** kernel parameter. This scheme is disabled by default in RHEL 7, and systemd's Predictable Network Interface feature is preferred.

Without the Predictable Network Interface feature, network interfaces retain their original ethX style names. With no OS function to ensure specific interfaces are given the same name at each boot event, device naming becomes unpredictable and is unacceptable for an OpenStack installation. Predictable interface naming uses udev rules to choose device names, in this order:

1. on-board devices with firmware-provided index numbers (example: eno1)
2. add-in devices with firmware-provided PCIe slot index numbers (example: ens1)
3. a device named using a physical connector location (example: enp2s0)
4. a device name incorporating the interface's MAC address (example: enx78e7d1ea46da)
5. finally, the classic, unpredictable kernel-native naming (example: eth0)

The only supported circumstances where it is safe to disable the Predictable Network Interface feature by setting **net.ifnames=0** are listed here:

- If the system only has a single interface and will never have more than a single interface.

- KVM guests exclusively using virtio-net type interfaces can safely set net.ifnames=0.
- If the system is custom configured for unique, non ethX style names using udev rules or the udev properties in the /usr/lib/udev/rules.d/60-net.rules file.

In this course, each node has multiple network interfaces, named using the ethX style, since the classroom is built with KVM guests using virtio-net. In this configuration, no naming order conflict would exist.

Using NetworkManager and the SDN Controllers

NetworkManager is a dynamic network control and configuration system that attempts to keep network devices and connections up and active when they are available. NetworkManager can be used to configure Ethernet, wireless, and other network interface types like network aliases, static routes, DNS information and VPN connections, plus many connection-specific parameters. In enterprise computing, NetworkManager is an indispensable tool for managing and scaling complex networks.

In cloud computing, Software-Defined Networking (SDN) controllers configure and manage both the interfaces of the physical infrastructure, and the virtual interfaces and devices of the virtualized cloud environment, providing a complete view of the network topology and its current state. For those devices, NetworkManager's actions would be in conflict with SDN controllers.

Therefore, to ensure NetworkManager does not take control of these devices, the OpenStack installation process adds the **NM_CONTROLLED="no"** parameter to every **/etc/sysconfig/network-scripts/ifcfg-*** file managed by the SDN controllers.

The NetworkManager service still controls the bridge used by Docker and should not be disabled.

NETWORK ANALYSIS COMMANDS

Virtual networking can be described as layering an upper network built only of virtualized software components, on top of the Linux operating systems virtual network for managing virtual machines and containers, on top of the physical interfaces and devices that allow multiple systems in an OpenStack cloud infrastructure to communicate together. Although software defined networking is logical and learn-able, it has multiple layers of complexity. Still, the network tools used to query virtual networks, bridges and interfaces are the same tools used to query legacy networks in enterprise computing.

Identifying Device Types Using the ethtool Command

You can use the **ethtool** command to identify device types. For example, the driver, or the bus type can be retrieved by running the following command:

```
[root@demo ~]# ethtool -i eth0
driver: virtio_net
version:
firmware-version:
bus-info: virtio0
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

All network interface files are located in the **/sys/class/net** directory, which provides the active settings for every interface.

Managing Interfaces Using the ip Command

The **ip** command is the primary tool for managing network devices in Linux. The **ip link** subcommand can modify layer 2 settings, such as the MAC address, the promiscuous mode, or bring the interface up or down. The **ip address** subcommand can modify layer 3 settings, such as the IP address as well as the IP netmask. The following table lists some of the available options for the **ip** command.

ip Command Options

COMMAND	PURPOSE
ip link	Lists all network interfaces.
ip link set device up down	Enables or disables a network link.
ip link set device promisc on off	Enables or disables promiscuous mode.
ip address show dev device	Lists the details of a given device.
ip address flush dev device	Removes the network settings of a given device.
ip address add IPADDR/IPMASK dev device	Adds an IP address to a given interface.
ip address del IPADDR/IPMASK dev device	Deletes an IP address from a given interface.

Introducing the ping Command

The **ping** command is a Linux tool that administrators can use to send ICMP packets to network hosts to elicit a response from the solicited host. The **ping** command supports the following options:

- **-c**: Stops after sending *count ECHO_REQUEST* packets. With this option, the **ping** command waits for the count of **ECHO_REPLY** packets to match the value, until the timeout is reached.

```
[root@demo ~]# ping -c 1 172.25.250.254
```

- **-i**: Sets an interval to wait, in seconds, between sending each packet. By default, the command pauses for one second between each packet.

```
[root@demo ~]# ping -i 3 172.25.250.254
```

- **-Q**: Sets the bits for the *Quality of service (QoS)* in the ICMP datagram, which is a unit of transfer. This option accepts either decimal or hexadecimal values.

```
[root@demo ~]# ping -Q 50 172.25.250.254
```

Managing Interfaces Using Linux Networking Tools

The following steps outline the process of adding an ephemeral IP address to a network interface, verifying the IP address, and monitoring the network connectivity using networking tools.

1. Log in as a user with administrative privileges on a Linux server. Ensure that the *iproute* package is installed.
2. Use **ip link** to check the link status for all network interfaces. To check the link status for a particular interface, use **ip link show DEVICE_NAME**.
3. Use **ip address add** to add an IP address to a given interface. The new IP address is not persistent and will be lost when rebooting the server.
4. The **ip address show** command lists the details of a particular network device.
5. To bring up a network interface, use the **ip link set** command with the **up** option. Using this command with the **down** option brings an interface down.

Adding a Persistent IP Address to a Network Interface

Administrators can configure network interfaces to use the same configuration after every reboot. The following steps outline the process to add a persistent IP address to a network interface.

1. As a user with administrative privileges on a Linux server, view the existing network interfaces. Use the **ip link** command to check the link status of these interfaces.
2. Create a file under **/etc/sysconfig/network-scripts** with the name **ifcfg-device name**. The device name for the network interface will vary depending on the types of devices installed in the system as recognized by systemd's predictable interface naming rules. For virtual instances in this classroom's VMs, the Ethernet interfaces are named **ethX**. The **ifcfg-device name** file defines the method and value of the IP address assigned to the interface.
3. Restart or reload the network service.
4. Using the **ip** command, retrieve the information for the network interface.



REFERENCES

Further information is available in the *Networking Guide* for Red Hat Enterprise Linux 7 at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

Further information is available in the *Red Hat Knowledgebase: Overview of systemd for RHEL 7* at
<https://access.redhat.com/articles/754933>

Further information is available in the *Network Functions Virtualization Planning Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_planning_and_configuration_guide/

Further information is available in the **man** command.

- **ip(8)**
- **ip-address(8)**
- **ip-link(8)**

MANAGING NETWORK INTERFACES

In this exercise, you will use various networking tools and assign IP addresses to the network interfaces attached to an instance.

OUTCOMES

You should be able to:

- Install and run **ethtool**, **tcpdump**, **ip**, and **ping**.
- Assign persistent IP addresses to the network interfaces attached to an instance.

Confirm the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as the password. On **workstation**, run the **lab manage-interfaces setup** command. The lab script launches an instance named **finance-server1** with three network interfaces, and creates the identity environment file for the **developer1** user.

```
[student@workstation ~]$ lab manage-interfaces setup
```

- ▶ 1. On **workstation**, use the command line to source the **/home/student/developer1-finance-rc** identity environment file to provide access to OpenStack services for the **developer1** user.

```
[student@workstation ~]$ source ~/developer1-finance-rc  
[student@workstation ~(developer1-finance)]$
```

- ▶ 2. Fetch the console URL for the **finance-server1** instance.

```
[student@workstation ~(developer1-finance)]$ openstack console url \  
show finance-server1  
+-----+  
| Field | Value |  
+-----+  
| type | novnc |  
| url | http://172.25.250.50:6080/vnc_auto.html?token=9c45...c761 |  
+-----+
```

- ▶ 3. Use this URL to connect to the **finance-server1** instance console from the Firefox browser.

```
[student@workstation ~(developer1-finance)]$ firefox \  
http://172.25.250.50:6080/vnc_auto.html?token=9c45...c761
```

- 4. Use **root** as the user name and **redhat** as the password to log in to the instance.

```
small image
finance-server1 login: root
Password: redhat
[root@finance-server1 ~]#
```

- 5. Using the **ip a** command verify the network interfaces attached to the **finance-server1** instance.

```
[root@finance-server1 ~]# 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 1450 qdisc pfifo_fast state UP qlen 1000
    link/ether fa:16:3e:ac:1a:92 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.7/24 brd 192.168.0.255 scope global dynamic eth0
        valid_lft 77736sec preferred_lft 77736sec
        inet6 fe80::f816:3eff:feac:1a92/64 scope link
            valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether fa:16:3e:9a:70:48 brd ff:ff:ff:ff:ff:ff
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether fa:16:3e:81:40:70 brd ff:ff:ff:ff:ff:ff
```

- 6. Use the **ethtool** command to view the network driver for the **eth0** interface.

```
[root@finance-server1 ~]# ethtool -i eth0
driver: virtio_net
version: 1.0.0
firmware-version:
expansion-rom-version:
bus-info: 0000:00:03.0
supports-statistics: no
supports-test: no
supports-eeprom-access: no
supports-register-dump: no
supports-priv-flags: no
```

- 7. Create and configure the **ifcfg-eth1** configuration file. Configure the boot protocol to be DHCP. The gateway is **192.168.20.1**. Reboot the **finance-server1** instance.

```
[root@finance-server1 ~]# vim /etc/sysconfig/network-scripts/ifcfg-eth1
DEVICE=eth1
TYPE=Ethernet
BOOTPROTO=dhcp
GATEWAY=192.168.20.1
```

```
[root@finance-server1 ~]# reboot
```

- 8. Log in to the instance and use the **tcpdump** command to monitor the packets received and sent through the **eth1** interface. Run the **tcpdump** command as a background process. The description for the contents of the packets gathered by the command must be written to the **tcpdump-eth1.txt** file.

```
[root@finance-server1 ~]# nohup tcpdump -n -i eth1 > tcpdump-eth1.txt 2>/dev/null
&
[1] 10783
```

- 9. Verify the IP address assigned to the **eth1** interface.

```
[root@finance-server1 ~]# ip a show dev eth1
3: eth1: >BROADCAST,MULTICAST,UP,LOWER_UP< mtu 1500 qdisc pfifo_fast state UP
    group default qlen 1000
        link/ether fa:16:3e:11:e8:50 brd ff:ff:ff:ff:ff:ff
        inet 192.168.20.11/24 scope global eth1
            valid_lft forever preferred_lft forever
```

- 10. Use **ping** to send ICMP requests to the **192.168.20.1** IP address from the **eth1** interface.

```
[root@finance-server1 ~]# ping -I eth1 -c 3 192.168.20.1
PING 192.168.20.1 (192.168.20.1) from 192.168.20.11 eth1: 56(84) bytes of data.
64 bytes from 192.168.20.1: icmp_seq=1 ttl=64 time=3.66 ms
64 bytes from 192.168.20.1: icmp_seq=2 ttl=64 time=1.02 ms
64 bytes from 192.168.20.1: icmp_seq=3 ttl=64 time=1.17 ms

--- 192.168.20.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 1.027/1.958/3.669/1.211 ms
```

- 11. Stop the **tcpdump** command. Analyze the packet information captured by the **tcpdump** command for the **eth1** interface.

11.1. Stop the background process running the **tcpdump**.

```
[root@finance-server1 ~]# jobs
[1]+  Running    nohup tcpdump -n -i eth1 > tcpdump-eth1.txt 2> /dev/null &
```

Note the job number specified in the square brackets and use **kill** to stop the process.

```
[root@finance-server1 ~]# kill %1
```

11.2. View the packet information in the **tcpdump-eth1.txt** file. The packet information shows the ICMP request originating from the **192.168.20.11** IP address, and the ICMP reply received from the **192.168.20.1** gateway IP address. Also notice that the interface fetches the MAC address of the gateway by sending an ARP (Address

Resolution Protocol) request. The response to the ARP request is a reply from the gateway with its MAC address.

```
[root@finance-server1 ~]# cat tcpdump-eth1.txt
05:16:09.653429 ARP, Request who-has 192.168.20.1 tell 192.168.20.11, length 28
05:16:09.657206 ARP, Reply 192.168.20.1 is-at fa:16:3e:e1:d8:64, length 28
05:16:09.657215 IP 192.168.20.11 > 192.168.20.1: ICMP echo request, ...
05:16:09.658732 IP 192.168.20.1 > 192.168.20.11: ICMP echo reply, ...
05:16:10.655636 IP 192.168.20.11 > 192.168.20.1: ICMP echo request, ...
05:16:10.657148 IP 192.168.20.1 > 192.168.20.11: ICMP echo reply, ...
05:16:11.657208 IP 192.168.20.11 > 192.168.20.1: ICMP echo request, ...
05:16:11.658503 IP 192.168.20.1 > 192.168.20.11: ICMP echo reply, ...
05:16:14.665080 ARP, Request who-has 192.168.20.11 tell 192.168.20.1, length 28
05:16:14.665097 ARP, Reply 192.168.20.11 is-at fa:16:3e:73:e9:10, length 28
```

- 12. Log out of the instance and close the browser tab for the instance console.

```
[root@finance-server1 ~]# exit
```

Cleanup

On **workstation**, run the **lab manage-interfaces cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab manage-interfaces cleanup
```

This concludes the guided exercise.

IMPLEMENTING LINUX BRIDGES

OBJECTIVE

After completing this section, students should be able to implement Linux bridges.

INTRODUCTION TO LINUX BRIDGES

A bridge is a device that combines two or more network segments into a single logical network, such as a single IP subnet. On a multi-interface system, such as one with multiple NICs or a virtualization host with virtual machines with network interfaces, the default behavior is to not automatically facilitate traffic between those interfaces. An administrator would create and connect a bridge to allow interfaces to connect and pass traffic.

A network bridge is a device that connects two Ethernet segments, but it is not a router. Routing occurs at layer 3 of the IP protocol, bridges operate at layer 2, the Ethernet layer, passing packets between ports on the bridge based on software-configured access rules. When packets traverse a bridge, the packet movement is based on the Ethernet address rather than an IP address. Packets are passed on a Layer 2 basis; all upper protocols are compatible with network bridges if they operate in Ethernet networks.

There are various implementations of bridges. Linux comes with its own implementation, managed by the **brctl** command. OpenStack uses Open vSwitch bridges to connect and filter traffic. Open vSwitch includes the following features:

- Support of the 802.1Q standard, for VLAN support.
- Multicast snooping, which allows for intelligent routing of traffic, instead of flooding all ports in a VLAN.
- Fine-grained control of *Quality of Service (QoS)* traffic.
- Support for multiple tunneling protocols, such as VXLAN and *Stateless Transport Tunneling (STT)*.
- Support for remote configuration protocol using the C and Python programming languages.
- Support for Kernel user-space forwarding.

MANAGING INTERFACES WITH THE BRCTL COMMAND

You can use the **brctl** command to manage Linux bridges. A Linux bridge, provided by a kernel module, is an emulated switch that connects physical and virtual interfaces. The **bridge** kernel module loads when the first Linux bridge is declared. Because bridges are a Kernel implementation, the network traffic management takes place in the kernel's memory space.

The following table lists some of the available options provided by the **brctl** command:

Using the brctl Command

COMMAND	PURPOSE
brctl addbr BRIDGE	Adds a bridge
brctl delbr BRIDGE	Deletes a bridge
brctl addif BRIDGE DEVICE	Adds a network interface to a bridge
brctl delif BRIDGE DEVICE	Deletes an interface from a bridge

COMMAND	PURPOSE
brctl show	Lists all available bridges
brctl showmacs BRIDGE	Lists the MAC addresses used by a bridge

Managing Bridge Interfaces

When administrators create bridges with the **brctl** command, they are immediately available, although not persistent across reboots. The following list describes some bridge management techniques:

- To create a bridge, use the **addbr** argument.

```
[root@demo ~]# brctl addbr br-demo
```

- To clear the IP allocation of a network device before enslaving it to a bridge, flush the device and then use the **addif** argument.

```
[root@demo ~]# ip address flush eth3
[root@demo ~]# brctl addif br-demo eth3
```

- To assign an IP address to a bridge, use the **ip** command.

```
[root@demo ~]# ip address add IP dev br-demo
```

- To access the details of a bridge, use the **show** subcommand.

```
[root@demo ~]# brctl show br-demo
bridge name bridge id      STP enabled interfaces
br-demo      8000.52540001000c  no      eth3
```

- To list the MAC addresses of all ports in a bridge, use the **showmacs** subcommand.

```
[root@demo ~]# brctl showmacs br-demo
port no mac addr      is local?    ageing timer
1 52:54:00:01:00:0c   yes          0.00
```

- To ensure that the kernel module is loaded, use the **lsmod** command.

```
[root@demo ~]# lsmod |grep bridge
bridge                  115385  0
stp                     12976   1 bridge
llc                     14552   2 stp,bridge
```

- To bring the bridge up, use the **ip** command.

```
[root@demo ~]# ip link set dev br-demo up
```

Managing Persistent Bridges

The network configuration files can be used to create persistent bridges; that is, bridges that are recreated when the system starts. Persistent bridges are defined the same way as network devices,

by creating a file named **/etc/sysconfig/network-scripts/ifcfg-BRIDGE**. However, to declare a network device as a bridge, the **TYPE=Bridge** entry must be set in the configuration file. The file only defines the bridge, not the devices attached to it. If you do not specify a MAC address for the bridge, it will be automatically generated.

The following example shows the configuration files used to create a persistent **br-demo** bridge and to define the **eth3** network device as a port for that bridge.

```
[root@demo ~]# cat /etc/sysconfig/network-scripts/ifcfg-br-demo
DEVICE=br-demo
TYPE=Bridge
IPADDR=172.25.249.100
PREFIX=24
ONBOOT=yes

[root@demo ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth3
DEVICE=eth3
BRIDGE=br-demo
ONBOOT=yes
```



IMPORTANT

The original *net-tools* package of network management tools, such as netstat, ifconfig, and ifup/ifdown, has been deprecated for almost a decade in favor of the *iproute2* package of newer tools, which includes the **ip** command set and more. To manage Linux bridges, use the new commands as in the following example:

```
[root@demo ~]# ip link set down dev BRIDGE
```

Creating Linux Bridge Interfaces Using the brctl Command

Bridges created with **brctl** are immediately available, but are not persistent across reboots. The following steps outline the process for creating Linux bridges using **brctl**.

1. As a user with administrative privileges, ensure that the *bridge-utils* package is installed.
2. To create a bridge, use the **brctl addbr** command.
3. Use **brctl addif** to add a network interface as a port to the bridge.
4. Assign an IP address to the Linux bridge using the **ip address add** command. The device name passed as an argument to the command is the bridge name.
5. To verify the bridge and interfaces are correctly configured, use the **brctl show** command.
6. Activate the bridge using the **ip link set dev bridge name up** command.

Managing Persistent Linux Bridges

The following steps outline the process for creating a persistent Linux bridge.

1. As a user with administrative privileges, create a file in the **/etc/sysconfig/network-scripts** directory. The bridge name and the corresponding file name containing the bridge configuration is defined by the user. The file defines the IP address assigned to the bridge and sets the **TYPE** to **Bridge**.
2. To add the network interface as a port to the Linux bridge, edit the network interface configuration file under **/etc/sysconfig/network-scripts**.

3. Restart or reload the network service.
4. Reboot the server to verify the configuration is persistent. (Optional)



REFERENCES

Further information is available in the *Networking Guide* for Red Hat Enterprise Linux 7 at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

Further information is available in the brctl(8) man page.

- brctl(8)

IMPLEMENTING LINUX BRIDGES

In this exercise, you will create a Linux bridge and assign a static IP address to the bridge.

OUTCOMES

You should be able to:

- Create a Linux bridge.
- Assign a static IP address to the bridge.

Confirm the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as the password. On **workstation**, run the **lab linuxbridges setup** command.

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

- 1. From **workstation** log in to **compute0** as the **heat-admin** user. Using **sudo -i** become root on the **compute0** node.

```
[student@workstation ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]#
```

- 2. View the information of the **eth3** interface. Create a new bridge called **br-linux1** and display its information.

2.1. Using the **ip link show** command view the details of the MAC address of **eth3**.

```
[root@compute0 ~]# ip link show eth3
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP mode
DEFAULT group default qlen 1000
link/ether 52:54:00:03:00:02 brd ff:ff:ff:ff:ff:ff
```

2.2. Using the **brctl** command create a bridge called **br-linux1**.

```
[root@compute0 ~]# brctl addbr br-linux1
[root@compute0 ~]#
```

2.3. Using the **ip link show** command view the details of the MAC address of the **br-linux1** bridge.

```
[root@compute0 ~]# ip link show br-linux1
63: br-linux1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
group default qlen 1000
link/ether fe:48:da:d5:b0:29 brd ff:ff:ff:ff:ff:ff
```

- 3. Add the **eth3** interface to **br-linux1**. View the details of the **eth3** interface and the **br-linux1** bridge.

- 3.1. Using the **brctl** command with the **addif** option to attach the **eth3** device to the bridge.

```
[root@compute0 ~]# brctl addif br-linux1 eth3  
[root@compute0 ~]#
```

- 3.2. Using the **ip link show** command view the details of the MAC address for both **eth3** and **br-linux1**.

```
[root@compute0 ~]# ip link show eth3  
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master br-  
linux1 state UP mode DEFAULT group default qlen 1000  
    link/ether 52:54:00:03:00:02 brd ff:ff:ff:ff:ff:ff  
[root@compute0 ~]# ip link show br-linux1  
63: br-linux1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT  
group default qlen 1000  
    link/ether 52:54:00:03:00:02 brd ff:ff:ff:ff:ff:ff
```

Note that the MAC addresses for **eth3** and **br-linux1** match. The linux bridge **br-linux1** can now reply on behalf of the linux interface **eth3**.

- 4. Using the **ip address add** command add the **172.25.249.33** IP address to the **br-linux1** bridge.

```
[root@compute0 ~]# ip address add 172.25.249.33 dev br-linux1  
[root@compute0 ~]#
```

- 5. Using the **brctl show** command, access the details of the **br-linux1** bridge.

```
[root@compute0 ~]# brctl show br-linux1  
bridge name      bridge id           STP enabled  interfaces  
br-linux1        8000.525400030002  no          eth3
```

- 6. Using the **brctl** command with the **showmacs** option list the MAC addresses of all ports in the **br-linux1** bridge.

```
[root@compute0 ~]# brctl showmacs br-linux1  
port no mac addr            is local?   ageing timer  
  1      52:54:00:03:00:02   yes        0.00  
  1      52:54:00:03:00:02   yes        0.00
```

- 7. Bring up the bridge using the **ip link** command.

```
[root@compute0 ~]# ip link set dev br-linux1 up  
[root@compute0 ~]#
```

- 8. Using the **ip addr** command view the device state of the **br-linux1** bridge.

```
[root@compute0 ~]# ip addr show dev br-linux1
63: br-linux1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    group default qlen 1000
        link/ether 52:54:00:03:00:02 brd ff:ff:ff:ff:ff:ff
        inet 172.25.249.33/24 scope global br-linux1
            valid_lft forever preferred_lft forever
        inet6 fe80::5054:ff:fe03:2/64 scope link
            valid_lft forever preferred_lft forever
```

- 9. Log out of the **compute0** node.

```
[root@compute0 ~]# exit
[heat-admin@compute0 ~]$ exit
[student@workstation ~]$
```

Cleanup

On **workstation**, run the **lab linuxbridges cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab linuxbridges cleanup
```

This concludes the guided exercise.

IMPLEMENTING OPEN VSWITCH BRIDGES

OBJECTIVES

After completing this section, students should be able to implement Open vSwitch (OVS) bridges.

INTRODUCTION TO OPEN VSWITCH (OVS)

Although Linux bridges emulate network bridges, they primarily only support the basic Ethernet standards, but not hardware vendor specific extensions. The Open vSwitch project, which supports multiple network protocols and standards, provides extra features designed for software defined networks (SDN). Open vSwitch supports network protocols such as VLAN, VXLAN, the OpenFlow standard, the sFlow protocol, as well as the Cisco RSPAN protocol. Open vSwitch can operate in a distributed configuration, with centralized controllers that provision the network bridges. Linux bridges and Open vSwitch bridges can coexist on a system.

Open vSwitch uses a database to store both the network configuration and network state. This database can be saved and migrated between different hosts. Open vSwitch can act as a software-based switch running on a virtual machine's hypervisor, and as a control stack for dedicated switching hardware. As a result, it has been ported to multiple virtualization platforms and switching chipsets. In OpenStack, OVS bridges are used instead of Linux bridges to provide the extra processing functionality required to interface with neutron and other components that normal Linux bridges are not capable of handling.

The following is a list of some of the Open vSwitch features:

- Support for the NetFlow, sFlow, IPFIX, SPAN, and RSPAN network protocols
- Support for *Link Aggregation Control Protocol* (LACP)
- Support for 802.1Q IEEE standard (VLAN), for network partitioning and trunking
- Support for *Spanning Tree Protocol* (STP)
- Control of Quality of service (QoS) for applications, users, and data flows
- Bonding of network interfaces with load balancing
- Full IPv6 support
- Remote configuration via C and Python bindings
- Implementation of packet forwarding in either the kernel space or the user space

Using the ovs-vsctl Command

You can use the **ovs-vsctl** command to manage Open vSwitch bridges. This command is available in the *openvswitch* package. Bridges and ports can be created, viewed, and destroyed. You can also use **ovs-vsctl** to connect the bridges to external SDN controllers. Bridges created with **ovs-vsctl** are immediately active and persistent across reboots. You can also configure Open vSwitch via the **ifcfg** files and the **ip** command.

The following table lists some common uses of the **ovs-vsctl** command.

Using the ovs-vsctl Command

COMMAND	PURPOSE
ovs-vsctl show	Displays an overview of Open vSwitch database contents

COMMAND	PURPOSE
ovs-vsctl add-br <i>BRIDGE</i>	Defines a new bridge named <i>BRIDGE</i>
ovs-vsctl del-br <i>BRIDGE</i>	Deletes the <i>BRIDGE</i> bridge and its ports
ovs-vsctl list-br	Lists all bridges
ovs-vsctl add-port <i>BRIDGE PORT</i>	Adds the <i>PORT</i> network device to the <i>BRIDGE</i>
ovs-vsctl del-port <i>BRIDGE PORT</i>	Removes the <i>PORT</i> network device from the <i>BRIDGE</i>
ovs-vsctl list-ifaces <i>BRIDGE</i>	Lists all interfaces on <i>BRIDGE</i>
ovs-vsctl iface-to-br <i>INTERFACE</i>	Displays the bridge that contains the specified interface
ovs-vsctl list-ports <i>BRIDGE</i>	Displays ports attached to the <i>BRIDGE</i>

Managing Open vSwitch Interfaces

The following lists some of the current options for managing Open vSwitch devices.

- To create a bridge, use the **add-br** command.

```
[root@demo ~]# ovs-vsctl add-br br-ovs-demo
```

- To specify an IP address to a bridge, use the **ip** command.

```
[root@demo ~]# ip address add 10.0.10.15/24 dev br-ovs-demo
```

- To add a network device to a bridge, use the **add-port** command.

```
[root@demo ~]# ovs-vsctl add-port br-ovs-demo eth1
```

- To bring up a bridge, use the **ip** command.

```
[root@demo ~]# ip link set dev br-ovs-demo up
```

- To review the content of the Open vSwitch database, use the **show** command.

```
[root@demo ~]# ovs-vsctl show
```

Open vSwitch Flow Mode

Open vSwitch can operate like a legacy network switch (by maintaining a MAC table) or it can operate in flow mode. In flow mode, various rules, created based on packet-matching criteria, determine the correct ports for outgoing traffic. Rules are created and injected into Open vSwitch bridges.

Flow mode is similar to how switch and router network processors manage traffic. Rules are created to match most any layer 2, layer 3, and layer 4 packet header field. When a rule matches, the packet is forwarded to a physical or virtual port, dropped, or rewritten for further processing by other rule sets. Rules are sorted by priority, and every frame is checked against the rules, starting

with the highest priority. If a frame matches a critiera set, then no further rule processing is done. All frames are first processed by **table 0**, before being optionally redirected to other tables. If there are no rules that match, the default policy is to drop the frame.

To determine the mode of a bridge, use the **dump-flows** command.

```
[root@demo ~]# ovs-ofctl dump-flows br-int
NXST_FLOW reply (xid=0x4):
  cookie=0x0, duration=677422.600s, table=0, n_packets=1214, n_bytes=114387,
  idle_age=617,
  cookie=0x0, duration=677230.905s, table=0, n_packets=1346, n_bytes=120003,
  idle_age=617,
  cookie=0x0, duration=677422.064s, table=0, n_packets=850, n_bytes=57920,
  idle_age=65534,
  cookie=0x0, duration=677422.562s, table=23, n_packets=0, n_bytes=0,
  idle_age=65534,

  hard_age=65534, priority=1 actions=NORMAL
  hard_age=65534, priority=3,in_port=1,dl_vlan=1 actions=mod_vlan_vid:1,NORMAL
  hard_age=65534, priority=2,in_port=1 actions=drop
  hard_age=65534, priority=0 actions=drop
```

INTRODUCTION TO OPENFLOW

Routers maintain path information about the flow of packets related to network traffic passing through them. This information is organized in *flow tables*, which are created and stored in the network hardware. Before 2011, access to this information was only allowed by the local software operating on the router. The OpenFlow standard was established to allow access to the flow tables from remote systems.

OpenFlow is not a comprehensive networking standard such as the Ethernet and TCP/IP standards. Rather, it requires a centralized controller to handle the network flow. A controller can be used for basic operations, such as monitoring the broadcasting traffic and building a MAC table, as well as advanced software-driven operations, such as monitoring the creation of virtual machines, and provisioning the network flows instantly across the network. Many projects provide SDN-based controllers to provide such features. For example, the OpenDaylight platform, an open source initiative, is an SDN designed to use the OpenFlow standard with OpenStack.

Using the **ovs-ofctl** Command

The **ovs-ofctl** command-line interface is a tool for monitoring and managing OpenFlow switches. This tool can be used to show the current state of an OVS bridge, including features, configuration, and table entries. It is compatible with OpenFlow switches created by Open vSwitch or OpenFlow.

The following is a list of some of the current options for managing OpenFlow switches with the **ovs-ofctl** command.

- To display an overview of the OpenFlow database, use the **show** command.

```
[root@demo ~]# ovs-ofctl show
```

- To retrieve the status of network ports, use the **dump-ports-desc** command.

```
[root@demo ~]# ovs-ofctl dump-ports-desc
```

- To display the entries for all flows, use the **dump-flows** command.

```
[root@demo ~]# ovs-ofctl dump-flows
```

Implementing OVS Bridges Using the **ovs-vsctl** Command

OVS bridges created with the **ovs-vsctl** tool are immediately available and persist across reboots. The following steps outline the process for creating an Open vSwitch bridge using the **ovs-vsctl** command. Because the **ip address add** command is used to add ephemeral IP addresses, the connectivity to the bridge is lost on reboot.

1. To create an OVS bridge, use the **ovs-vsctl add-br** command.
2. Use **ovs-vsctl list bridge** to list bridge attributes in the OVS database. To set a bridge attribute, use the **ovs-vsctl set bridge BRIDGE** command. (Optional)
For example, the **ovs-vsctl set bridge demo-br other-config:hwaddr='00:00:00:00:00:01'** command sets the hardware address for the **demo-br** bridge.
3. Use the **ovs-vsctl add-port** command to add the interface to the bridge.
4. Assign an IP address to the OVS bridge. Use the **ip address add** command to assign an IP address to the bridge device.
5. Bring up the OVS bridge device using the **ip link set dev BRIDGE up** command.
6. Confirm the bridge is configured correctly using the **ovs-vsctl show** command. List the ports attached to the bridge using the **ovs-vsctl list-ports** command.

Monitoring the OpenFlow Switches Using **ovs-ofctl**

The **ovs-ofctl** tool displays the current state of an OVS bridge, including features, configuration, and table entries. The following procedure describes how to monitor the Open vSwitch bridge using **ovs-ofctl**.

1. For an existing Open vSwitch bridge, use **ovs-ofctl show** to list OpenFlow features and port descriptions.
2. The **ovs-ofctl dump-ports-desc** command is useful for viewing port connectivity. The command is useful for detecting errors in the network interfaces of a bridge.
3. Use the **ovs-ofctl dump-flows** command to print all the flow entries from the OpenFlow bridge.



REFERENCES

Further information is available in the *Configure Bridge Mappings in OpenStack Networking* solution at

https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux_openstack_platform/7/html/networking_guide/bridge-mappings

Further information is available in the following **man** pages:

- `ovs-vsctl(8)`
- `ovs-ofctl(8)`
- `ovs-dpctl(8)`
- `ovs-vswitchd(8)`
- `ovs-appctl(8)`

IMPLEMENTING OPEN VSWITCH BRIDGES

In this exercise, you will manage Open vSwitch bridges.

OUTCOMES

You should be able to:

- Create an Open vSwitch bridge and add a port.
- Use the Open vSwitch tools to manage the bridge.

Confirm that the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as the password. On **workstation**, run the **lab ovsbridges setup** command.

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

- ▶ 1. On **workstation**, log in to the **compute0** node as the **heat-admin** user. Using **sudo -i** become root on the **compute0** node.

```
[student@workstation ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]#
```

- ▶ 2. Using the **rpm -q** command confirm that the **openvswitch** package is installed.

```
[root@compute0 ~]# rpm -q openvswitch
openvswitch-2.9.0-19.el7fdp.1.x86_64
```

- ▶ 3. Using the **systemctl** command check the status of the **openvswitch** service.

```
[root@compute0 ~]# systemctl status openvswitch.service
● openvswitch.service - Open vSwitch
   Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled; vendor
   preset: disabled)
     Active: active (exited) since Mon 2018-06-11 08:23:23 UTC; 13min left
       Process: 1021 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
      Main PID: 1021 (code=exited, status=0/SUCCESS)
        Tasks: 0
       Memory: 0B
        CGroup: /system.slice/openvswitch.service

Jun 11 08:23:23 compute0 systemd[1]: Starting Open vSwitch...
Jun 11 08:23:23 compute0 systemd[1]: Started Open vSwitch.
```

- 4. View the information of the **eth4** interface. Create a new OVS bridge called **br-ovs1** and display its information.

4.1. Using the **ip link show** command view the details of the MAC address of **eth4**.

```
[root@compute0 ~]# ip link show eth4
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
    system state UP mode DEFAULT group default qlen 1000
        link/ether 52:54:00:04:00:02 brd ff:ff:ff:ff:ff:ff
```

4.2. Using the **ovs-vsctl** command create an Open vSwitch bridge called **br-ovs1**.

```
[root@compute0 ~]# ovs-vsctl add-br br-ovs1
[root@compute0 ~]#
```

4.3. Using the **ip link show** command view the details of the MAC address of **br-ovs1**.

```
[root@compute0 ~]# ip link show br-ovs1
64: br-ovs1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
    group default qlen 1000
        link/ether ca:8a:5c:4e:10:48 brd ff:ff:ff:ff:ff:ff
```

- 5. Add the **eth4** interface to **br-ovs1**. Assign a specific MAC address to the **br-ovs1** bridge. View the details of the **eth4** interface and the **br-ovs1** bridge.

5.1. Using the **ovs-vsctl** command with the **add-port** option add the **eth4** network device to the **br-ovs1** bridge.

```
[root@compute0 ~]# ovs-vsctl add-port br-ovs1 eth4
[root@compute0 ~]#
```

5.2. Using the **ovs-vsctl set** command assign the MAC address **52:54:00:04:00:01** to the **br-ovs1** bridge.

```
[root@compute0 ~]# ovs-vsctl set bridge br-ovs1 \
other-config:hwaddr=\"52:54:00:04:00:01\"
[root@compute0 ~]#
```

5.3. Using the **ip link show** command view the details of the MAC address of the **eth4** network interface and the **br-ovs1** bridge.

```
[root@compute0 ~]# ip link show eth4
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
    system state UP mode DEFAULT group default qlen 1000
        link/ether 52:54:00:04:00:02 brd ff:ff:ff:ff:ff:ff
[root@compute0 ~]# ip link show br-ovs1
64: br-ovs1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN mode DEFAULT
    group default qlen 1000
        link/ether 52:54:00:04:00:01 brd ff:ff:ff:ff:ff:ff
```

Note that unlike linux bridges OVS bridges do not automatically emulate the linux interface.

- ▶ 6. Using the **ip** command specify the **172.25.249.55** IP address for the **br-ovs1**.

```
[root@compute0 ~]# ip address add 172.25.249.55/24 dev br-ovs1
[root@compute0 ~]#
```

- ▶ 7. Using the **ip** command bring up the **br-ovs1** bridge.

```
[root@compute0 ~]# ip link set dev br-ovs1 up
[root@compute0 ~]#
```

- ▶ 8. Using the **ovs-vsctl** command with the **show** option review the content of the Open vSwitch database.

```
[root@compute0 ~]# ovs-vsctl show | grep -A3 br-ovs1
    Bridge "br-ovs1"
        Port "br-ovs1"
            Interface "br-ovs1"
            type: internal
        Port "eth4"
            Interface "eth4"
```

- ▶ 9. To list the ports of the **br-ovs1** bridge use the **ovs-vsctl** command with the **list-ports** option.

```
[root@compute0 ~]# ovs-vsctl list-ports br-ovs1
eth4
```

- ▶ 10. Using the **ip addr show** command view the device state of the **br-ovs1** bridge.

```
[root@compute0 ~]# ip addr show br-ovs1
64: br-ovs1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue
    state UNKNOWN group default qlen 1000
        link/ether 52:54:00:04:00:01 brd ff:ff:ff:ff:ff:ff
        inet6 fe80::5054:ff:fe04:2/64 scope link
            valid_lft forever preferred_lft forever
```

- ▶ 11. Log out of the **compute0** node.

```
[root@compute0 ~]# exit
[heat-admin@compute0 ~]$ exit
[student@workstation ~]$
```

Cleanup

On **workstation**, run the **lab ovsbridges cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab ovsbridges cleanup
```

This concludes the guided exercise.

MANAGING LINUX NETWORKS

PERFORMANCE CHECKLIST

In this lab, you will manage Linux interfaces, create a Linux bridge and an Open vSwitch bridge. You will attach these bridges to **eth3** and **eth4** of the **compute0** node.

OUTCOMES

You should be able to manage Linux interfaces and create Linux and Open vSwitch bridges.

Confirm that the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab networks-lab setup** script that launches two instances named **production-server1** and **production-server2**.

```
[student@workstation ~]$ lab networks-lab setup
```

1. On **workstation**, source the **/home/student/operator1-production-rc** identity environment file for the **operator1** user.
2. Using the **openstack server list** command, retrieve the IP addresses of **production-server1** and **production-server2**. Log in to **production-server1**, view the ARP table. Ping **production-server2** from **production-server1**. View the ARP table on **production-server1** and **production-server2**.
3. From **workstation** log into **compute0** as the **heat-admin** user. Using the **sudo -i** command become root on the **compute0** node.
4. On **compute0** create a Linux bridge named **br-linux1**. View the device state for **br-linux1**. Attach the **eth3** network interface to the bridge. Configure the **br-linux1** bridge with the IP address **172.25.249.100** and a netmask of **255.255.255.0**.
5. Ensure that the kernel module is loaded. Bring up the bridge **br-linux1**.
6. On **compute0** create an Open vSwitch bridge named **br-ovs1**. Attach the **eth4** network interface as a port to the bridge. Configure the **br-ovs1** bridge with the IP address **172.25.249.110** and a netmask of **255.255.255.0**.
7. Bring up the bridge interface. Verify the settings for the **br-ovs1** OVS bridge and the **eth4** network interface port added to it.
8. Log out of **compute0**.

```
[root@compute0 ~]# exit  
[heat-admin@compute0 ~]$ exit  
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab networks-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab networks-lab grade
```

Cleanup

On **workstation**, run the **lab networks-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab networks-lab cleanup
```

This concludes the lab.

MANAGING LINUX NETWORKS

PERFORMANCE CHECKLIST

In this lab, you will manage Linux interfaces, create a Linux bridge and an Open vSwitch bridge. You will attach these bridges to **eth3** and **eth4** of the **compute0** node.

OUTCOMES

You should be able to manage Linux interfaces and create Linux and Open vSwitch bridges.

Confirm that the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab networks-lab setup** script that launches two instances named **production-server1** and **production-server2**.

```
[student@workstation ~]$ lab networks-lab setup
```

1. On **workstation**, source the **/home/student/operator1-production-rc** identity environment file for the **operator1** user.

```
[student@workstation ~]$ source ~/operator1-production-rc  
[student@workstation ~(operator1-production)]$
```

2. Using the **openstack server list** command, retrieve the IP addresses of **production-server1** and **production-server2**. Log in to **production-server1**, view the ARP table. Ping **production-server2** from **production-server1**. View the ARP table on **production-server1** and **production-server2**.
 - 2.1. Use the **openstack server list** command to list the servers in the **production** project.

```
[student@workstation ~(operator1-production)]$ openstack server list \  
-c Name -c Networks -c Status  
+-----+-----+-----+  
| Name | Status | Networks |  
+-----+-----+-----+  
| production-server2 | ACTIVE | production-network1=192.168.1.Y |  
| production-server1 | ACTIVE | production-network1=192.168.1.X |  
+-----+-----+-----+
```

- 2.2. Use the **openstack console url show** command to retrieve the console URL for **production-server1**.

```
[student@workstation ~(operator1-production)]$ openstack console url \  
show production-server1  
+-----+-----+  
| Field | Value |
```

```
+-----+-----+
| type | novnc
| url  | http://172.25.250.50:6080/vnc_auto.html?token=8212...1873 |
+-----+-----+
```

- 2.3. Use this URL to connect to the **production-server1** instance console from the Firefox browser. Log in as **root** with **redhat** as the password.

```
[student@workstation ~] $ firefox \
http://172.25.250.50:6080/vnc_auto.html?token=8212...1873
```

- 2.4. Review the ARP table.

```
[root@production-server1 ~] # arp
Address      HWtype  HWaddress          Flags Mask   Iface
192.168.1.2  ether    fa:16:3e:c5:3d:86  C      eth0
gateway      ether    fa:16:3e:0b:7e:32  C      eth0
```

- 2.5. Ping **production-server2** from **production-server1**. Review the ARP table again.

```
[root@production-server1 ~] # ping -c 3 192.168.1.Y
PING 192.168.1.Y (192.168.1.Y) 56(84) bytes of data.
64 bytes from 192.168.1.Y: icmp_seq=1 ttl=64 time=1.17 ms
64 bytes from 192.168.1.Y: icmp_seq=2 ttl=64 time=0.335 ms
64 bytes from 192.168.1.Y: icmp_seq=3 ttl=64 time=0.458 ms

--- 192.168.1.Y ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.335/0.656/1.176/0.371 ms
[root@production-server1 ~] # arp
Address      HWtype  HWaddress          Flags Mask   Iface
192.168.1.Y  ether    fa:16:3e:15:9f:f1  C      eth0
192.168.1.2  ether    fa:16:3e:c5:3d:86  C      eth0
gateway      ether    fa:16:3e:0b:7e:32  C      eth0
```

- 2.6. Log in to **production-server2** to display the ARP table. For that, from **workstation**, use the **openstack console url show** command to retrieve the console URL for **production-server2**.

```
[student@workstation ~] $ openstack console url \
show production-server2
+-----+-----+
| Field | Value
+-----+-----+
| type  | novnc
| url   | http://172.25.250.50:6080/vnc_auto.html?token=1234...1865 |
```

Use this URL to connect to the **production-server2** instance console from the Firefox browser. Log in as **root** with **redhat** as the password.

On **production-server2**, display the ARP table. You will see an entry for **production-server1** is present due to the ping traffic.

```
[root@production-server2 ~]$ arp
Address          HWtype  HWaddress          Flags Mask   Iface
gateway         ether    fa:16:3e:0b:7e:32  C      eth0
192.168.1.X   ether    fa:16:3e:a3:19:85  C      eth0
192.168.1.2     ether    fa:16:3e:c5:3d:86  C      eth0
```

2.7. Log out of both **production-server1** and **production-server2**

```
[root@production-server1 ~]# logout
```

```
[root@production-server2 ~]$ logout
```

3. From **workstation** log into **compute0** as the **heat-admin** user. Using the **sudo -i** command become root on the **compute0** node.

```
[student@workstation ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]#
```

4. On **compute0** create a Linux bridge named **br-linux1**. View the device state for **br-linux1**. Attach the **eth3** network interface to the bridge. Configure the **br-linux1** bridge with the IP address **172.25.249.100** and a netmask of **255.255.255.0**.

- 4.1. On **compute0** ensure that the *bridge-utils* package is installed.

```
[root@compute0 ~]# rpm -q bridge-utils
bridge-utils-1.5-9.el7.x86_64
```

- 4.2. Using the **brctl** command create a bridge called **br-linux1**.

```
[root@compute0 ~]# brctl addbr br-linux1
[root@compute0 ~]#
```

- 4.3. View the device state of the **br-linux1** bridge.

```
[root@compute0 ~]# ip addr show dev br-linux1
34: br-linux1: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
    qlen 1000
        link/ether 7a:14:5b:0e:69:c4 brd ff:ff:ff:ff:ff:ff
```

- 4.4. Using **brctl** with the **addif** option attach the **eth3** device to the bridge.

```
[root@compute0 ~]# brctl addif br-linux1 eth3
```

```
[root@compute0 ~]#
```

- 4.5. Using the **ip address add** command add the **172.25.249.100/24** IP address to **br-linux1**.

```
[root@compute0 ~]# ip address add 172.25.249.100/24 dev br-linux1
[root@compute0 ~]#
```

5. Ensure that the kernel module is loaded. Bring up the bridge **br-linux1**

- 5.1. Using **lsmod** ensure that the kernel module is loaded.

```
[root@compute0 ~]# lsmod | grep bridge
bridge                  146976  1 br_netfilter
stp                     12976   1 bridge
llc                     14552   2 stp,bridge
```

- 5.2. Using **ip link** bring up the bridge **br-linux1**.

```
[root@compute0 ~]# ip link set dev br-linux1 up
[root@compute0 ~]#
```

- 5.3. Using **ip addr** view the device state for **br-linux1**.

```
[root@compute0 ~]# ip addr show dev br-linux1
34: br-linux1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
    group default qlen 1000
        link/ether 7a:14:5b:0e:69:c4 brd ff:ff:ff:ff:ff:ff
        inet 172.25.249.100/24 scope global br-linux1
            valid_lft forever preferred_lft forever
        inet6 fe80::7a14:5bff:fe0e:69c4/64 scope link
            valid_lft forever preferred_lft forever
```

6. On **compute0** create an Open vSwitch bridge named **br-ovs1**. Attach the **eth4** network interface as a port to the bridge. Configure the **br-ovs1** bridge with the IP address **172.25.249.110** and a netmask of **255.255.255.0**.

- 6.1. On **compute0**, as the **root** user, ensure that the *openvswitch* package is installed.

```
[root@compute0 ~]# rpm -q openvswitch
openvswitch-2.9.0-19.el7fdp.1.x86_64
```

- 6.2. Check the status of the **openvswitch** service.

```
[root@compute0 ~]# systemctl status openvswitch.service
● openvswitch.service - Open vSwitch
  Loaded: loaded (/usr/lib/systemd/system/openvswitch.service; enabled; vendor
  preset: disabled)
  Active: active (exited) since Tue 2018-06-12 07:52:49 UTC; 1h 19min ago
    Process: 1020 ExecStart=/bin/true (code=exited, status=0/SUCCESS)
   Main PID: 1020 (code=exited, status=0/SUCCESS)
     Tasks: 0
    Memory: 0B
      CGroup: /system.slice/openvswitch.service

Jun 12 07:52:49 compute0 systemd[1]: Starting Open vSwitch...
```

```
Jun 12 07:52:49 compute0 systemd[1]: Started Open vSwitch.
```

- 6.3. Using the **ovs-vsctl** command create an Open vSwitch bridge called **br-ovs1**.

```
[root@compute0 ~]# ovs-vsctl add-br br-ovs1
[root@compute0 ~]#
```

- 6.4. Using the **ip** command specify the **172.25.249.110/24** IP address for the **br-ovs1**.

```
[root@compute0 ~]# ip address add 172.25.249.110/24 dev br-ovs1
[root@compute0 ~]#
```

- 6.5. Using **ovs-vsctl** with the **add-port** option add the **eth4** network device to the **br-ovs1** bridge.

```
[root@compute0 ~]# ovs-vsctl add-port br-ovs1 eth4
[root@compute0 ~]#
```

7. Bring up the bridge interface. Verify the settings for the **br-ovs1** OVS bridge and the **eth4** network interface port added to it.

- 7.1. Using the **ip** command bring up the **br-ovs1** bridge.

```
[root@compute0 ~]# ip link set dev br-ovs1 up
[root@compute0 ~]#
```

- 7.2. Review the IP and the MAC addresses assigned to the **br-ovs1** OVS bridge.

```
[root@compute0 ~]# ip a
...output omitted...
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    group default qlen 1000
        link/ether 52:54:00:04:00:02 brd ff:ff:ff:ff:ff:ff
        inet6 fe80::5054:ff:fe04:2/64 scope link
            valid_lft forever preferred_lft forever
...output omitted...
41: br-ovs1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        link/ether f2:a2:e5:35:61:49 brd ff:ff:ff:ff:ff:ff
        inet 172.25.249.110/24 scope global br-ovs1
            valid_lft forever preferred_lft forever
        inet6 fe80::f0a2:e5ff:fe35:6149/64 scope link
            valid_lft forever preferred_lft forever
```

- 7.3. Use **ovs-vsctl** to confirm that the bridge and port are correctly configured.

```
[root@compute0 ~]# ovs-vsctl show | grep -A2 br-ovs1
    Bridge "br-ovs1"
        Port "eth4"
            Interface "eth4"
        Port "br-ovs1"
            Interface "br-ovs1"
                type: internal
    Bridge br-tun
```

8. Log out of **compute0**.

```
[root@compute0 ~]# exit  
[heat-admin@compute0 ~]$ exit  
[student@workstation ~]$
```

Evaluation

On **workstation**, run the **lab networks-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab networks-lab grade
```

Cleanup

On **workstation**, run the **lab networks-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab networks-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The TCP/IP standards follow a layer network model specified in **RFC1122**: Application layer, Transport layer, Internet layer, and Link layer.
- Administrators can persistently configure network interfaces. These settings are defined in the configuration files located in **/etc/sysconfig/network-scripts/**. The configuration files are named **ifcfg-*NAME***, where *NAME* refers to the name of the device that the configuration file controls.
- A network bridge is a device that connects two Ethernet segments. When packets traverse a bridge, they are forwarded based on their Ethernet address, rather than their IP address.
- The Open vSwitch project, which supports multiple network protocols and standards, aims to provide extra features for virtualization networks.

CHAPTER 5

PREPARING TO DEPLOY AN INSTANCE

GOAL

Manage images, flavors, and private networks in preparation for launching an instance.

OBJECTIVES

- Manage software profiles (images).
- Manage hardware profiles (flavors).
- Manage private networks.

SECTIONS

- Uploading Images (and Guided Exercise)
- Developing Flavors (and Guided Exercise)
- Managing Private Networks (and Guided Exercise)

LAB

- Lab: Preparing to Deploy an Instance

UPLOADING IMAGES

OBJECTIVE

After completing this section, students should be able to manage software profiles (images).

INTRODUCTION TO IMAGES

In Red Hat OpenStack Platform, an *image* is a file containing a virtual disk installed with a bootable operating system. Images are managed by the Image Service (glance). An image is one of the fundamental requirements for deploying instances.

Images can be uploaded by any user. The following image formats are currently supported on this version of Red Hat OpenStack Platform:

Image Formats

FORMAT	DESCRIPTION
raw	An unstructured disk image format.
vhd	A common disk format used by virtual machine monitors from VMware, Xen, Microsoft, VirtualBox, and others.
vmdk	Another common disk format supported by many common virtual machine monitors.
vdi	A disk format supported by VirtualBox virtual machine monitor and the QEMU emulator.
iso	An archive format for the data contents of an optical disc (for example, CD-ROM).
qcow2	A disk format supported by the QEMU emulator that can expand dynamically and supports the copy-on-write feature.
aki	An Amazon kernel image, supported by Amazon EC2.
ari	An Amazon ramdisk image, supported by Amazon EC2.
ami	An Amazon machine image, supported by Amazon EC2.
ploop	A disk format supported by the QEMU emulator. It can expand dynamically and supports Copy on Write.
ova	An OVF package in a tarfile.
docker	A docker container format.

Images are managed using either the OpenStack dashboard or the **openstack** CLI command. The OpenStack dashboard image management is found under Project → Compute → Images. The **openstack** CLI command for image management is **openstack image** command.

MANAGING AN IMAGE

Each instance requires an image to create its virtual disk. This image contains the minimum required software, including a bootable OS. This image is stored and served by the Image service, which manages the catalog of images. Depending on configurable attributes, images are available to some or all users, to create and deploy new instances in the Red Hat OpenStack Platform environment.

When deploying an instance, the compute service obtains the requested image from the image service and copies it to the libvirt cache on the compute node where the instance deployment is scheduled. Additional instances launched using the same image and scheduled to the same compute node, will use the cached image, avoiding another copy from the image service. During initial instance deployment, the virtual disk is resized to the size requested in the flavor.



NOTE

When creating images, it is important to keep them small and generic. Deployment-time customizations are performed by cloud-init or configuration management tools, as discussed in a later chapter. Limiting the image size reduces network copying time to the compute node, and results in faster initial deployments.

The Image service is usually deployed to a controller node along with other Red Hat OpenStack Platform services, separate from the compute nodes. The Image service's image store may be located on the controller node's disk in small OpenStack installations. Transferring large images from the controller node to compute nodes can impact controller node performance. For this reason, production installations either configure the compute node image cache as shared storage across all compute nodes, or configure Red Hat Ceph Storage, a scalable and distributed storage system, as the default image store. Installing Red Hat Ceph Storage with OpenStack is accepted as the industry standard configuration.

IMAGE PROPERTIES

An image has some metadata attributes associated with it. These attributes can be used by other OpenStack services to verify image requirements and settings.

One of the most-used settings for an image is its visibility. By default, during an image's creation, its visibility is set to *shared*, allowing the image to be accessible by the owner, as well as other users within the same project. Other image visibility settings include *private*, *community*, and *public*. Private visibility limits image access to the owner while public visibility allows access to all users. Community visibility allows all users access to the image but only lists the image in the default image list of certain users.

Images also have settings associated with them that define their requirements. For example, a minimum disk size and amount of RAM can be defined for an image. Finally, an image can be also configured as *protected* so it cannot be removed accidentally.

IMAGE LIMITS

An instance uses both an image as the template for its virtual system disk, and a flavor that defines the hardware resources used to deploy that instance. If an image includes minimum requirements for disk and memory size, those requirements must be met by the flavor settings, else the deployment request is rejected. If an image does not include minimum requirement settings, it can be deployed using almost any flavor, but if the flavor sizing is insufficient for the image to successfully boot, the deployment will fail.

To avoid issues with instance deployment, images should always be set with correct minimum sizes for proper operation. Similarly, the flavor used must specify disk and memory sizes that are equal to or larger than specified by the image. If minimum requirements are not matched by the flavor,

the CLI will issue an error message. In the dashboard, while deploying an instance using the wizard, flavors that do not meet the disk and memory requirements for the selected imaged, are disabled. If no flavors qualify to deploy this image, an error message is displayed.

Managing an Image Using the Dashboard

The following steps outline the process for managing an image using the Dashboard.

1. Open the Dashboard in a web browser, and log in as a user associated with a project. Navigate to Project → Compute → Images and click Create Image. Enter the Image Name.
2. Click Browse and navigate to the image file that you want to upload. Select the image format in the Format field.
3. Select **Private** in the Visibility field to configure the image as private. Click Create Image.
4. Log in to the Dashboard as an administrative user in the same project as the user in the previous steps.
5. Navigate to Project → Compute → Images and click Create Image. Enter the Image Name.
6. Click Browse and navigate to the image file that you want to upload. Select the image format in the Format field.
7. Select **Public** in the Visibility field to configure the image as public. Click Create Image.
8. Refresh the web page to refresh the image status. Select Edit Image in the menu displayed for the image you just created in the previous step, as an OpenStack administrator. Enter values for the Minimum Disk(GB) and Minimum RAM(MB) fields, then click Update Image.
9. Click Delete Image in the menu displayed for the previous image to delete the private image.

Managing an Image Using the OpenStack CLI

The following steps outline the process for managing an image using the OpenStack unified CLI.

1. Source the identity environment file to load the credentials for the user.
2. Create an image using the **openstack image create** command. By default images are configured as private.
3. Source the identity environment file to load the credentials for a user with administrator privileges. To create a public image, the user must have administrator privileges.
4. Create an image using the **openstack image create** command. Configure it as a public image using the **--public** option.
5. Configure the previous image with a minimum disk size and RAM using the **openstack image set** command with the **--min-disk** and **--min-ram** options.
6. Delete the private image using the **openstack image delete** command.



REFERENCES

Further information is available in the Image Service section of the *Instances and Images Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/instances_and_images_guide/

UPLOADING IMAGES

In this exercise, you will manage images using the dashboard and the unified CLI. You will create and delete images, and modify the image settings, including the visibility, minimum disk capacity, and protection.

OUTCOMES

You should be able to:

- Create and manage images using the dashboard.
- Create and manage images using the unified CLI tools.
- Delete an image.

Ensure that the **workstation** and overcloud virtual machines are started.

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

On **workstation**, run **lab prepinternal-images setup**, which verifies that OpenStack services are running and the resources created in previous sections are available.

```
[student@workstation ~]$ lab prepinternal-images setup
```

- ▶ 1. On **workstation**, open a terminal and download the **osp-small.qcow2** image file from <http://materials.example.com/osp-small.qcow2> to the **/home/student/Downloads** directory.

```
[student@workstation ~]$ wget -P ~/Downloads \
http://materials.example.com/osp-small.qcow2
...output omitted...
2018-06-13 06:27:18 (240 MB/s) - '/home/student/Downloads/osp-small.qcow2' saved
[842661888/842661888]
```

- ▶ 2. Open Firefox and browse to <http://dashboard.overcloud.example.com>.
- ▶ 3. Log in to the dashboard as the **developer1** user with password, **redhat**. Use the **osp-small.qcow2** image you just downloaded to create an image named **rhel7-small** using the QCOW2 format and the default settings.
- Log in to the dashboard using **developer1** as the user name and **redhat** as the password.
 - Navigate to Project → Compute → Images and click +Create Image. Enter the following details for the image:
 - Image Name: **rhel7-small**
 - File: **/home/student/Downloads/osp-small.qcow2**
 - Format: QCOW2 - QEMU Emulator

- Protected: No

Click the Create Image button to create the image. Wait until the Status field value for the **rhel7-small** image is **Active**. You may need to refresh the web page for the Status field to be updated.

- 4. Set the **rhel7-small** image status to **protected**, and set the minimum disk size to 10 GB. Log out of the dashboard.
- 4.1. Click the arrow next to the Launch button for the row labeled **rhel7-small**. Select Edit Image to edit the image.
Select **Yes** in the Protected field. Enter **10** in the Minimum Disk(GB) field. Click Update Image.
 - 4.2. Log out of the dashboard.
- 5. Use the CLI to create a new image, named **rhel7-small2**, using the **developer1** user credentials. Create this image from the **osp-small.qcow2** image file previously downloaded and use the QCOW2 format.
- 5.1. On **workstation**, open a terminal. Source the **/home/student/developer1-finance-rc** environment file to export the **developer1** user credentials.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$
```

- 5.2. Create the **rhel7-small2** image using the **osp-small.qcow2** image file and the QCOW2 format.

```
[student@workstation ~(developer1-finance)]$ openstack image create \
--disk-format qcow2 \
--file ~/Downloads/osp-small.qcow2 \
rhel7-small2
+-----+
| Field      | Value
+-----+
| checksum   | d3bc2ff8b3533697a0acaacd5b28016d
| container_format | bare
| created_at | 2018-06-15T02:58:03Z
| disk_format | qcow2
| file       | /v2/images/3369dc9b-05f4-4455-9348-23d3a39f6344/file
| id         | 3369dc9b-05f4-4455-9348-23d3a39f6344
| min_disk   | 0
| min_ram   | 0
| name       | rhel7-small2
| owner      | cec9ea4041ff4aeb82236ea702e0e338
| properties | direct_url='...', locations='...'
| protected  | False
| schema     | /v2/schemas/image
| size       | 842661888
| status     | active
| tags       |
| updated_at | 2018-06-15T02:58:14Z
| virtual_size | None
| visibility | shared
+-----+
```

- 6. Set the **rhel7-small2** image status to **protected**, and set the minimum disk size to 10 GB.

```
[student@workstation ~ (developer1-finance)]$ openstack image set \
--protected \
--min-disk 10 \
rhel7-small2
```

- 7. Verify that the **rhel7-small2** image was correctly created by listing the available images, and showing the details for the **rhel7-small2** image.

- 7.1. List the available images and confirm the **Status** for the **rhel7-small2** image is **active**.

```
[student@workstation ~ (developer1-finance)]$ openstack image list
+-----+-----+-----+
| ID           | Name      | Status |
+-----+-----+-----+
| 30c17bad-ce14-4d0d-abb9-0f5b323adace | rhel7-small   | active  |
| 3369dc9b-05f4-4455-9348-23d3a39f6344 | rhel7-small2 | active  |
...output omitted...
```

- 7.2. Display the details of the **rhel7-small2** image, and verify that it was correctly updated. Confirm the value for the **min_disk** field is **10**, and the value for the **protected** field is **True**.

```
[student@workstation ~ (developer1-finance)]$ openstack image show rhel7-small2
+-----+-----+
| Field      | Value
+-----+-----+
| checksum    | d3bc2ff8b3533697a0acaacd5b28016d
| container_format | bare
| created_at  | 2018-06-15T02:58:03Z
| disk_format | qcow2
| file        | /v2/images/3369dc9b-05f4-4455-9348-23d3a39f6344/file
| id          | 3369dc9b-05f4-4455-9348-23d3a39f6344
| min_disk    | 10
| min_ram    | 0
| name        | rhel7-small2
| owner       | cec9ea4041ff4aeb82236ea702e0e338
| properties  | direct_url='...', locations='...'
| protected   | True
| schema      | /v2/schemas/image
| size        | 842661888
| status      | active
| tags        |
| updated_at  | 2018-06-15T03:00:32Z
| virtual_size | None
| visibility   | shared
+-----+-----+
```

- 8. Delete the **rhel7-small2** image. You need to unprotect it before deleting.

8.1. Try to delete the **rhel7-small2** image without unprotecting it. It should return an error.

```
[student@workstation ~ (developer1-finance)]$ openstack image delete rhel7-small2
Failed to delete image with name or ID 'rhel7-small2': 403 Forbidden:
Image 3369dc9b-05f4-4455-9348-23d3a39f6344 is protected and cannot be deleted.
(HTTP 403)
Failed to delete 1 of 1 images.
```

8.2. Set the **rhel7-small2** image to be unprotected.

```
[student@workstation ~ (developer1-finance)]$ openstack image set \
--unprotected \
rhel7-small2
```

8.3. Delete the **rhel7-small2** image.

```
[student@workstation ~ (developer1-finance)]$ openstack image delete rhel7-small2
```

- 9. Using the **architect1** user credentials, which has administrative rights, make the **rhel7-small** image public. You must have administrative rights to make an image public. When done, verify that the **rhel7-small** image has been correctly updated.

9.1. Source the **~/architect1-finance-rc** file to load the **architect1** user credentials. This user has administrative rights.

```
[student@workstation ~ (developer1-finance)]$ source ~/architect1-finance-rc
[student@workstation ~ (architect1-finance)]$
```

9.2. Set the **rhel7-small** image to be public.

```
[student@workstation ~ (architect1-finance)]$ openstack image set \
--public \
rhel7-small
```

9.3. Verify that the **rhel7-small** image has been correctly updated.

```
[student@workstation ~ (architect1-finance)]$ openstack image show rhel7-small
+-----+-----+
| Field      | Value
+-----+-----+
| checksum    | d3bc2ff8b3533697a0acaacd5b28016d
| container_format | bare
| created_at   | 2018-06-15T02:54:47Z
| disk_format   | qcow2
| file         | /v2/images/30c17bad-ce14-4d0d-abb9-0f5b323adace/file
| id           | 30c17bad-ce14-4d0d-abb9-0f5b323adace
| min_disk     | 10
| min_ram      | 0
| name         | rhel7-small
| owner        | cec9ea4041ff4aeb82236ea702e0e338
| properties    | direct_url='...', locations='...'
| protected    | True
```

schema	/v2/schemas/image
size	842661888
status	active
tags	
updated_at	2018-06-15T03:30:21Z
virtual_size	None
visibility	public
+-----+	+-----+

Cleanup

On **workstation**, run the **lab prepinternal-images cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab prepinternal-images cleanup
```

This concludes the guided exercise.

DEVELOPING FLAVORS

OBJECTIVE

After completing this section, students should be able to manage hardware profiles (flavors).

INTRODUCTION TO FLAVORS

Flavors are hardware specification profiles for deploying instances. These specifications include the disk and memory size, and number of cores to be used for each instance deployed. Flavors can also specify sizes for additional ephemeral storage or a swap disk, plus metadata to restrict usage or to provide special project access.

MANAGING FLAVORS

You must specify a flavor to deploy an instance. The Compute service schedules the deployment by checking the compute nodes for sufficient available resources, as requested by the flavor. If no host (compute node) can be found with sufficient resources, the instance deployment fails. Any compute node with sufficient resources and an acceptable current load can be selected for this deployment. Scheduling algorithms in the Compute service define all the criteria used to prioritize compute nodes for the selected deployment.

Flavors are managed with the Dashboard or with the OpenStack unified CLI. The OpenStack dashboard flavor management is found under Admin → System → Flavors. This functionality is only available to users with administrator privileges. The **openstack** CLI command supports image management with the **openstack flavor** command.

Flavors can be customized so that hardware requirements meet user needs. In addition to system disk and memory size, and the number of VCPUs, other parameters can be defined, such as adding a swap disk or additional ephemeral disk.

Creating new flavors is restricted to users with administrative privileges, to avoid misuse of cloud resources. Red Hat OpenStack Platform provides the ability to restrict flavor access by project membership. Flavors can be restricted for the exclusive requirements and use of a specific project by creating the flavor as *private*. For example, flavors with large resource parameters could be restricted to only projects requiring large sizes. If a flavor with 256 GB of memory was made public, all projects have access to that flavor, and misuse of that flavor could quickly consume the cloud environment's resource capacity. Even the **admin** project must conform to flavor restrictions.

The Compute service uses role-based access policies to specify which users can access and how they can use resource objects. The policy definitions are defined in the Compute service **policy.json** file. Policy rules determine whether any API request sent to the compute service will be accepted. The policy evaluates both the user and the object requested to determine whether to grant access.

DEFINING STORAGE IN FLAVORS

Flavors define the virtual storage to be made available to an instance. There can be several storage resources associated with an instance, including the root disk, an ephemeral disk, and a swap disk. All of these storage types are *ephemeral*, meaning that these virtual disks will be discarded when the instance is terminated. These disk resources are backed by files located in the **/var/lib/nova/instances/** directory of the compute node where the instance has been deployed. When

the instance is created, a subdirectory is created using the instance ID as the name, to contain the base image files for the root, ephemeral, and swap disks.

Root Disk

The root disk contains the operating system for the instance, created using an image as a template. If flavor's root disk size is smaller than the image's minimum disk requirement the instance deployment will not be attempted. The Compute service also supports using persistent volumes as the root disk source, which is discussed in a later chapter in this course.

Ephemeral Disk

The ephemeral disk in a flavor defines an additional disk to be attached to the deployed instance. Like the root disk, the ephemeral disk is created as a disk device in the instance at the size set in the flavor. This disk is a raw device, requiring partitioning, formatting, filesystem creation, and mounting to a directory before it can be used. By default, the size of an ephemeral disk in a flavor is 0 GB, meaning that no disk is created. The **cloud-init** utility, discussed later in this course, automates these tasks during instance deployment.

Swap Disk

A swap disk defines additional storage that can be enabled as swap in the instance. Similar to the ephemeral disk, swap space is presented as a raw device in the instance. To use this device as swap, it needs to be declared and enabled as swap space.

OTHER PARAMETERS IN A FLAVOR

Red Hat OpenStack Platform administrators customize flavors for specific environments and use cases. For example, in an environment with suboptimal networking, flavors can implement instance bandwidth restrictions. When systems are limited to a maximum number of sockets, flavors can configure limits and preferences for sockets, cores, and threads. Disk quotas can be set to limit the maximum write rate per second for a user. Flavor customizations are implemented by **extra_specs** element.

Using the `extra_specs` element

The **extra_specs** flavor element is used to define free-form characteristics, providing flexibility beyond specifying memory, CPU, and disk specifications. The element uses key-value pairs that assist in scheduling the compute nodes for an instance deploy. Key-value flavor settings must match corresponding key-value settings on compute nodes. For example, to configure the maximum number of supported CPU sockets, use the **hw:cpu_max_sockets** key. The following is a list of keys provided by the **extra_specs** element:

hw:action

The action that configures support limits.

hw:NUMA_def

The definition of the NUMA topology for the instance.

hw:watchdog_action

Triggers an action if the instance somehow fails (or hangs).

hw_rng:action

The action that adds a random number generator device to an instance.

quota:option

A limit that is forced on the instance.

Disk tuning

The **extra_specs** element provides disk tuning options to customize performance. The following is a list of valid options:

Flavor parameters

PARAMETER	DESCRIPTION
disk_read_bytes_sec	Maximum disk reads in bytes per second.
disk_read_iops_sec	Maximum disk read I/O operations per second.
disk_write_bytes_sec	Maximum disk writes in bytes per second.
disk_write_iops_sec	Maximum disk write I/O operations per second.
disk_total_bytes_sec	Maximum disk total throughput limit in bytes per second.
disk_total_iops_sec	Maximum disk total I/O operations per second.

To implement disk I/O quotas, use the **openstack flavor set** command. For example, to set the maximum write speed for a VM instance to 10 MB per second using disk quotas, use the following command:

```
[user@demo ~]# openstack flavor set m2.small \
--property quota:disk_write_bytes_sec=10485760
```

To set the maximum read speed for a VM user to 10 MB per second using disk quotas, use the following command:

```
[user@demo ~]# openstack flavor set m2.small \
--property quota:disk_read_bytes_sec=10485760
```

Use the **openstack flavor show** command to view the flavor details, including the disk quotas.

```
[user@demo ~]# openstack flavor show -c name -c properties -f json m2.small
{
    "name": "m2.small",
    "properties": "quota:disk_read_bytes_sec='10485760',
    quota:disk_write_bytes_sec='10485760'"
}
```

Managing Flavors Using the Dashboard

The following steps outline the process for managing flavors using the Dashboard.

1. Open the Dashboard in a web browser, and log in as a user with administrator privileges. Navigate to Admin → Flavors and click Create Flavor. Complete the Name, VCPUs, RAM(MB), and Root Disk(GB) details, then click Create Flavor.
2. Select Edit Flavor in the menu displayed for the previous flavor. Enter values for the Ephemeral Disk(GB) and Swap Disk(MB) fields, then click Save.
3. Delete the flavor, selecting Delete Flavor in the menu displayed for that flavor.

Managing Flavors Using the OpenStack CLI

The following steps outline the process for managing flavors using the OpenStack unified CLI.

1. Source the identity environment file to load the credentials for a user with administrator privileges.
2. Create a flavor using the **openstack flavor create** command. Configure this flavor with a RAM size using the **--ram** option, and a root disk size using the **--disk** option.
3. Check the **id** of this flavor using the **openstack flavor show** command, and checking the **id** field value.
4. Delete the flavor using the **openstack flavor delete** command.
5. Create a flavor using the **openstack flavor create** command. Configure this flavor with the **id** of the previously deleted flavor, using the **--id** option. Configure this flavor with both the memory size and the root disk size from the deleted flavor, and specify an ephemeral disk size in GB using the **--ephemeral** option, and specify a swap disk size in GB using the **--swap** option.
6. Delete the flavor using the **openstack flavor delete** command.



REFERENCES

Additional information is available in the Manage Flavors section of the *Instances and Images Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/instances_and_images_guide/

DEVELOPING FLAVORS

In this exercise, you will manage flavors using the dashboard and the unified CLI. You will create and delete flavors, and modify the flavor settings, including the root disk, number of vCPUs, RAM, ephemeral disk, and swap disk.

OUTCOMES

You should be able to:

- Create a flavor using the dashboard.
- Create a flavor using the CLI.
- Delete a flavor.

Ensure that the **workstation** and overcloud virtual machines are started.

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

On **workstation**, run **lab prepinternal-flavors setup**, which verifies that OpenStack services are running and the resources created in previous sections are available.

```
[student@workstation ~]$ lab prepinternal-flavors setup
```

- ▶ 1. On **workstation**, open Firefox and browse to **http://dashboard.overcloud.example.com**.
- ▶ 2. Using the **architect1** user credentials in the dashboard, create a flavor named **small**. Configure this flavor with two vCPUs, 1024 MB of RAM and a 10 GB root disk.
 - 2.1. Log in to the dashboard using **architect1** as the user name and **redhat** as the password.
 - 2.2. Navigate to Admin → Compute → Flavors and click +Create Flavor. Enter the following details for the flavor:
 - Name: **small**
 - vCPUs: **2**
 - RAM (MB): **1024**
 - Root Disk (GB): **10**Click Create Flavor and create the flavor.

2.3. Log out of the Dashboard.

- ▶ 3. Create a flavor named **small-swap** using the **architect1** credentials in the CLI. Configure the flavor with two vCPUs, 1024 MB of RAM, a 10 GB root disk, a 2 GB ephemeral disk, and a 1024 MB swap disk.

- 3.1. Open a terminal on **workstation**.
- 3.2. Source the **/home/student/architect1-finance-rc** file to load the **architect1** user credentials.

```
[student@workstation ~]$ source ~/architect1-finance-rc
[student@workstation ~(architect1-finance)]$
```

- 3.3. Create a flavor named **small-swap**. Configure the flavor with two VCPUs, 1024 MB of RAM, a 10 GB root disk, a 2 GB ephemeral disk, and a 1024 MB swap disk.

```
[student@workstation ~(architect1-finance)]$ openstack flavor create \
--vcpus 2 \
--ram 1024 \
--disk 10 \
--ephemeral 2 \
--swap 1024 \
small-swap

+-----+-----+
| Field          | Value   |
+-----+-----+
| OS-FLV-DISABLED:disabled | False  |
| OS-FLV-EXT-DATA:ephemeral | 2      |
| disk           | 10     |
| id             | 053941f1-5607-408e-b8fd-b8e68b40abb0 |
| name           | small-swap |
| os-flavor-access:is_public | True   |
| properties      |        |
| ram            | 1024   |
| rxtx_factor    | 1.0    |
| swap           | 1024   |
| vcpus          | 2      |
+-----+-----+
```

- 3.4. Verify that the **small-swap** flavor has been created with the correct settings.

```
[student@workstation ~(architect1-finance)]$ openstack flavor show small-swap

+-----+-----+
| Field          | Value   |
+-----+-----+
| OS-FLV-DISABLED:disabled | False  |
| OS-FLV-EXT-DATA:ephemeral | 2      |
| access_project_ids     | None   |
| disk           | 10     |
| id             | cd725c48-9f68-4024-8712-3aa957c49ec8 |
| name           | small-swap |
| os-flavor-access:is_public | True   |
| properties      |        |
| ram            | 1024   |
| rxtx_factor    | 1.0    |
| swap           | 1024   |
| vcpus          | 2      |
+-----+-----+
```

► 4. Delete the **small** flavor.

4.1. Delete the **small** flavor.

```
[student@workstation ~](architect1-finance)]$ openstack flavor delete small
```

4.2. Verify that the **small** flavor has been correctly deleted.

```
[student@workstation ~](architect1-finance)]$ openstack flavor show small  
No flavor with a name or ID of 'small' exists.
```

Cleanup

On **workstation**, run the **lab prepinternal-flavors cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab prepinternal-flavors cleanup
```

This concludes the guided exercise.

MANAGING PRIVATE NETWORKS

OBJECTIVES

After completing this section, students should be able to manage private networks.

OPENSTACK NETWORKING

The OpenStack Networking Service (Neutron) is a virtual network service that provides a rich interface for defining network connectivity and IP addressing in the OpenStack environment. This service is based on plug-ins that give administrators the flexibility they require to leverage different networking technologies and strategies.

Administrators can configure rich network topologies by creating and configuring networks and subnets, and then instructing other OpenStack services, such as the Compute Service, to attach virtual devices to ports on these networks. In particular, the OpenStack Networking Service supports each project having multiple networks, and allows projects to choose their own IP addressing scheme, even if those IP addresses overlap with those used by other projects. This enables advanced cloud networking use cases, such as building multi-tiered web applications and allowing applications to be migrated to the cloud without changing IP addresses.

Introduction to OpenStack Networks

Projects are provided with their own networks by the OpenStack Networking Service. These networks are isolated using Virtual Local Area Network (VLAN) segregation (every project network is a network VLAN). Virtual Extensible LAN (VXLAN) can be used for tunneling between OpenStack nodes. Each network contains one or more subnetworks; network traffic is isolated within the network. Users can define multiple networks within their project.

The OpenStack Networking Service supports both external and internal networks. External networks provide connectivity outside of the cloud environment. A network is specified as external when created. Internal networks provide connectivity between nodes of the OpenStack cloud environment.

Networks implement isolation at the system level using namespaces. The *kernel* provided by the default Red Hat Enterprise Linux 7 installation includes network namespaces. Network namespaces cover network access with a layer of abstraction by virtualizing access to the network resources, such as ports and interfaces. The normal **ifconfig** and **ip** commands do not show IP addresses within these namespaces. The **iproute** command has also been updated to allow queries to these namespaces. To access a namespace, use the **ip netns** command using the UUID of the namespace.

By default, networks are created for use inside the project in which they are created. Networks can also be configured as shared networks, so that users in different projects can use them. When creating a new network, OpenStack default behavior creates an internal network, so additional steps must be taken to define an external network. If a Red Hat OpenStack Platform environment is configured to use multiple networking technologies as part of its infrastructure, for example VLAN, and VXLAN, a network can be created using one of them by specifying it as the network type.

In the CLI, network management is supported by the **openstack network create** command. The **--share** option defines a network as shared:

```
[user@demo ~]# openstack network create demonet --share
```

The **--external** option defines a network as external:

```
[user@demo ~(admin)]$ openstack network create demoextnet --external
```

The **--provider-network-type** defines which network backend to use when there are several ones defined:

```
[user@demo ~(admin)]$ openstack network create demonet --provider-network-type vxlan
```

Network management is supported in the dashboard in Project → Network → Networks.

Introduction to OpenStack Subnetworks

Instances are assigned subnetworks to allow for network connectivity. Subnetworks are configured for pre-existing networks and have a specific IP address scheme associated with them. Each project can have one or more subnetworks. This allows isolation between different systems that use the same network.

Subnetworks are created with an associated IP address range. This range can be specified either as a whole network, so that the complete network range will be used, or as an IP range, so only the IP addresses in that range are assigned to the instances deployed in that subnetwork. In both cases, an IP address is reserved for the subnetwork gateway.

By default, subnetworks are configured to use DHCP to assign IP addresses within its defined IP address range to the instances deployed on it. Subnetworks can be created without DHCP, which means that IP addresses must be configured manually in the instances. If DHCP is enabled when an instance is created in an existing subnet, the Compute service sends a request to the **neutron-server** service, which runs the OpenStack networking API. The Neutron server requests that an IP address be created. This request is managed by the OpenStack networking DHCP agent, which queries the instance of **dnsmasq** associated with the subnet where the instance has to be placed. The **dnsmasq** service returns an unused semi-random IP address in the subnetwork range to the Neutron DHCP agent so that it can send it back to the Neutron server. When an IP address has been assigned to the new instance, the Neutron server will request that Open vSwitch performs the required configuration to plug the new instance to the existing subnet, using the IP address previously assigned. This configuration is returned to the Neutron server using the RabbitMQ service, and the Neutron server returns it to the Compute service.

A DNS host name can be configured in a subnetwork. This DNS host name is configured on each instance deployed on the subnetwork as its default DNS host name.



IMPORTANT

In previous versions of Red Hat OpenStack Platform, instances using a floating IP address were configured to use the external network's DNS name server. Now, tenant subnets can be configured with different DNS name servers, which are passed to the instance when deployed on that subnet.

Subnetwork management is supported in the dashboard in the Project → Network → Networks section and in the CLI by the **openstack subnet** command.

USING THE OPEN VSWITCH PLUG-IN

The OpenStack networking service is composed of several agents. Each of those agents provides a specific function. These agents are listed in the following table.

AGENT	FEATURE
L3 agent	Provides external access to the OpenStack environment, including support for floating IP addresses, which allow external access to the instances running on the OpenStack environment.
DHCP agent	Provides support for subnets created using the OpenStack networking service. This agent starts a new DHCP service running in the namespace for each subnet being created. It provides the range of IP addresses this subnet has been created with.
Metadata agent	Provides information services for the instances that are being created to gain information about its configuration.
Open vSwitch agent	Manages the Open vSwitch plug-in for Neutron.

The OpenStack networking service uses the concept of a plug-in, which is a pluggable backend implementation of the OpenStack networking API. A plug-in can use a variety of technologies to implement the logical API requests. Some OpenStack networking plug-ins might use basic Linux VLANs and IP tables, while others might use more advanced technologies, such as L2-in-L3 tunneling or OpenFlow, to provide similar benefits. There are several plug-ins currently supported, such as Open vSwitch, the current default, or OpenDaylight. Open vSwitch uses the following networking elements to manage egress and ingress traffic to instances.

OPEN VSWITCH CONFIGURATION	
br-int	Provides both ingress and egress VLAN tagging for instance traffic.
br-ex	Provides an external bridge and connects to qg of the qrouter via a tap interface.
qg	Connects the router to the gateway.
qr	Connects the router to the integration bridge br-int .
qvo	Connects to the integration bridge.
qvb	Connects the firewall bridge to br-int via qvo .
qbr	Linux bridge to provide OpenStack security groups.

Ingress Packet Flow

An example of how the ingress traffic flows to an instance using the above networking elements is shown in the following list. This will be explained in more detail in a later chapter.

- Inbound packets first go to **eth0** on the physical system.
- From **eth0**, the packet is sent to the **br-ex** bridge.
- The packet is then sent to the integration bridge, **br-int**.
- When the packet arrives at **br-int**, the packet is modified by an OVS flow rule within the **br-int** bridge. The OVS flow rule adds VLAN tagging to the packet and forwards it to **qvo**.
- After **qvo** accepts the packet, it strips away the VLAN tag and forwards it to **qvb**.

- **qvb** receives the packet and enforces the security-group rules.
- If no rules impede the progress of the traffic, the packets arrive at the instance.

Managing Networks and Subnetworks Using the Dashboard

The following steps outline the process for managing networks and subnetworks using the Dashboard.

1. Open the Dashboard in a web browser, and log in as a user.
2. Navigate to Project → Network → Networks and click Create Network.
3. In the Network section, enter the network name in the Network Name field. Verify that the Create Subnet check box is selected. Click Next.
4. In the Subnet section, complete the Subnet Name and Network Address fields, then click Next.
5. In the Subnet Details section, complete the DNS Name Servers field, click Create and create both the network and subnetwork.
6. In the network list, click the network name under the Name column. Navigate to the Subnets tab.
7. Select Edit Subnet in the menu displayed for the previously created subnet.
8. In the Subnet section, click Next.
9. In the Subnet Details section, clear the Enable DHCP checkbox in order to disable DHCP in the subnetwork. Click Save.
10. In the menu displayed in the upper right of the network details web page, click Edit Network.
11. Change the network name in the Name field. Click Save Changes and update the network. Disregard the **Danger: There was an error submitting the form. Please try again.** popup window, which is displayed due to a known dashboard bug.
12. In the network list, click the network name under the Name column. Navigate to the Subnets tab.
13. Delete the subnetwork, selecting Delete Subnet in the menu displayed for that subnetwork. Click Delete Subnet again in the popup window in order to confirm.
14. In the menu displayed in the upper right of the network details web page, select Delete Network to delete the network. Click Delete Network again in the popup window to confirm.

Managing Networks and Subnetworks Using the OpenStack CLI

The following steps outline the process for managing networks and subnetworks using the OpenStack unified CLI.

1. Source the identity environment file to load the credentials for a user.
2. Create a network using the **openstack network create** command.
3. Create a subnetwork on the previously created network using the **openstack subnet create** command. Associate this subnetwork with the previously created network using the **--network** option. Configure the range in CIDR format for this subnetwork using the **--subnet-range** option. Configure the DNS name server for this subnetwork using the **--dns-nameserver** option.

4. Update the network name using the **openstack network set** command. Configure the new name for this network using the **--name** option.
5. Update the subnetwork allocation pool settings using the **openstack subnet set** command. Disable DHCP for this subnetwork using the **--no-dhcp** option.
6. Delete the subnetwork using the **openstack subnet delete** command.
7. Delete the network using the **openstack network delete** command.



REFERENCES

Additional information is available in the OpenStack Networking Concepts section of the *Networking Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

MANAGING PRIVATE NETWORKS

In this exercise, you will manage networks and subnetworks using the dashboard and the unified CLI. You will create and delete networks and subnetworks, update their settings, and rename them.

OUTCOMES

You should be able to:

- Create a network and a subnet using the dashboard.
- Create a network and a subnet using the CLI.
- Rename a network.
- Update a subnet.

Ensure that the **workstation** and overcloud virtual machines are started.

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

On **workstation**, run **lab prepinternal-networks setup**, which verifies that OpenStack services are running and the resources created in previous sections are available.

```
[student@workstation ~]$ lab prepinternal-networks setup
```

- ▶ 1. On **workstation**, open Firefox and browse to **http://dashboard.overcloud.example.com**.
- ▶ 2. Using the **developer1** user credentials in the dashboard, create a network named **finance-network2** and a subnet named **finance-subnet2** with the **192.168.2.0/24** network address. Log out of the dashboard when you have finished creating the network and subnet.
 - 2.1. Log in to the dashboard using **developer1** as the user name and **redhat** as the password.
 - 2.2. Navigate to Project → Network → Networks and click +Create Network. In the Network section, enter **finance-network2** in the Network Name field. Verify that the Create Subnet check box is selected. Click Next.
 - 2.3. In the Subnet section, enter **finance-subnet2** in the Subnet Name field. Enter **192.168.2.0/24** in the Network Address field. Click Next.
 - 2.4. In the Subnet Details section, review and accept the default settings. Then click Create.
 - 2.5. Log out of the dashboard.
- ▶ 3. Create a network named **finance-network4** using the **developer1** credentials in the CLI.
 - 3.1. On **workstation**, open a terminal.

- 3.2. Source the **/home/student/developer1-finance-rc** file to load the **developer1** user credentials.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$
```

- 3.3. Create a network named **finance-network4**.

```
[student@workstation ~(developer1-finance)]$ openstack network create \
finance-network4
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-15T05:33:30Z |
| description | |
| dns_domain | None |
| id | 02b7194c-eef0-4cac-867f-81dd2a3f33fa |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1450 |
| name | finance-network4 |
| port_security_enabled | True |
| project_id | cec9ea4041ff4aeb82236ea702e0e338 |
| provider:network_type | None |
| provider:physical_network | None |
| provider:segmentation_id | None |
| qos_policy_id | None |
| revision_number | 3 |
| router:external | Internal |
| segments | None |
| shared | False |
| status | ACTIVE |
| subnets | |
| tags | |
| updated_at | 2018-06-15T05:33:30Z |
+-----+-----+
```

- 4. Create a subnet named **finance-subnet4** within the **finance-network4** network. Configure this subnet to use the **192.168.4.0/24** range. Verify the settings of the subnet.
- 4.1. Create a subnet named **finance-subnet4** in the **finance-network4** network. Configure this subnet to use the **192.168.4.0/24** range.

```
[student@workstation ~(developer1-finance)]$ openstack subnet create \
--subnet-range 192.168.4.0/24 \
--network finance-network4 \
finance-subnet4
+-----+-----+
| Field | Value |
+-----+-----+
```

allocation_pools	192.168.4.2-192.168.4.254
cidr	192.168.4.0/24
created_at	2018-06-15T05:39:22Z
description	
dns_nameservers	
enable_dhcp	True
gateway_ip	192.168.4.1
host_routes	
id	3aa323aa-1695-4bdd-b088-999ada97006d
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	finance-subnet4
network_id	02b7194c-eef0-4cac-867f-81dd2a3f33fa
project_id	cec9ea4041ff4aeb82236ea702e0e338
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-06-15T05:39:22Z

4.2. Verify that the **network_id** field value for the **finance-subnet4** subnet is the **finance-network4** network ID.

[student@workstation ~](developer1-finance)]\$ openstack subnet show \
finance-subnet4
+-----+-----+
Field Value
+-----+-----+
allocation_pools 192.168.4.2-192.168.4.254
cidr 192.168.4.0/24
created_at 2018-06-15T05:39:22Z
description
dns_nameservers
enable_dhcp True
gateway_ip 192.168.4.1
host_routes
id 3aa323aa-1695-4bdd-b088-999ada97006d
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name finance-subnet4
network_id 02b7194c-eef0-4cac-867f-81dd2a3f33fa
project_id cec9ea4041ff4aeb82236ea702e0e338
revision_number 0
segment_id None
service_types
subnetpool_id None
tags
updated_at 2018-06-15T05:39:22Z

- ▶ 5. Change the name of the **finance-network2** network to **finance-network3**. Verify the network name.

5.1. Change the name of the **finance-network2** network to **finance-network3**.

```
[student@workstation ~ (developer1-finance)]$ openstack network set \
--name finance-network3 \
finance-network2
```

5.2. Verify that the **finance-network2** network name has been correctly changed to **finance-network3**.

```
[student@workstation ~ (developer1-finance)]$ openstack network list
+-----+-----+-----+
| ID      | Name          | Subnets    |
+-----+-----+-----+
...output omitted...
| de90...2648 | finance-network3 | 7841...4f47 |
...output omitted...
+-----+-----+-----+
```

- ▶ 6. Update the **finance-subnet2** subnet to disable DHCP. Verify the change.

6.1. Update the **finance-subnet2** subnet to disable DHCP.

```
[student@workstation ~ (developer1-finance)]$ openstack subnet set \
--no-dhcp \
finance-subnet2
```

6.2. Verify that the **finance-subnet2** subnet has been correctly updated.

```
[student@workstation ~ (developer1-finance)]$ openstack subnet show \
finance-subnet2
+-----+-----+
| Field        | Value          |
+-----+-----+
| allocation_pools | 192.168.2.2-192.168.2.254 |
| cidr         | 192.168.2.0/24 |
| created_at   | 2018-06-15T05:30:55Z |
| description   |                   |
| dns_nameservers |                   |
| enable_dhcp   | False           |
| gateway_ip    | 192.168.2.1   |
| host_routes   |                   |
| id            | 7841e286-950f-4b5e-a2fe-1aa6776d4f47 |
| ip_version    | 4               |
| ipv6_address_mode | None           |
| ipv6_ra_mode   | None           |
| name          | finance-subnet2 |
| network_id    | de906011-f182-40d1-8878-233fd3bb2648 |
| project_id    | cec9ea4041ff4aeb82236ea702e0e338 |
| revision_number | 1              |
| segment_id    | None           |
| service_types  |                   |
| subnetpool_id | None           |
```

tags		
updated_at	2018-06-15T05:50:14Z	

- 7. Delete the **finance-subnet2** subnet and ensure that the subnet has been correctly deleted.

7.1. Delete the **finance-subnet2** subnet.

```
[student@workstation ~ (developer1-finance)]$ openstack subnet delete \
finance-subnet2
```

7.2. Verify that the **finance-subnet2** subnet has been correctly deleted.

```
[student@workstation ~ (developer1-finance)]$ openstack subnet show \
finance-subnet2
```

```
No Subnet found for finance-subnet2
```

- 8. Delete the **finance-network3** network and ensure that the network has been correctly deleted.

8.1. Delete the **finance-network3** network.

```
[student@workstation ~ (developer1-finance)]$ openstack network delete \
finance-network3
```

8.2. Verify that the **finance-network3** network has been correctly deleted.

```
[student@workstation ~ (developer1-finance)]$ openstack network show \
finance-network3
```

```
Error while executing command: No Network found for finance-network3
```

Cleanup

On **workstation**, run the **lab_prepinternal-networks cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab_prepinternal-networks cleanup
```

This concludes the guided exercise.

PREPARING TO DEPLOY AN INSTANCE

PERFORMANCE CHECKLIST

In this lab, you will create the resources required to launch an instance, including an image, a project network and subnetwork, and a flavor.

OUTCOMES

You should be able to:

- Create an image, and configure it as public.
- Create a project network and a subnetwork.
- Create a flavor.

Ensure that the **workstation** and overcloud virtual machines are started.

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

On **workstation**, run **lab prepinternal-lab setup**, which verifies that OpenStack services are running and the resources created in previous sections are available. The lab script creates the credentials files on **workstation** in **/home/student/architect1-production-rc** and **/home/student/operator1-production-rc**.

```
[student@workstation ~]$ lab prepinternal-lab setup
```

If you need administrative privileges, use the **architect1** account (password: **redhat**). The credentials file can be found on **workstation** as **/home/student/architect1-production-rc**.

1. Use the **http://materials.example.com/osp-web.qcow2** file to create an image named **rhel7-web** using the **operator1** user credentials. Configure this image to be **protected** and to use 10 GB as the minimum disk size, and 2048 MB as the minimum amount of RAM.
2. Create a network named **production-network2** using the **operator1** user credentials, and a subnetwork on it named **production-subnet2**. Use the **192.168.2.0/24** range and the **172.25.250.254** DNS name server for the **production-subnet2** subnetwork.
3. Create a flavor named **default-extra-disk**. Configure this flavor with a 10 GB root disk, 2048 MB of RAM, two vCPUs, a 5 GB ephemeral disk, a 1024 MB swap disk, and use 42 as the ID of the flavor.
4. Configure the **rhel7-web** image to be **public**.

Evaluation

On **workstation**, run the **lab prepinternal-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab prepinternal-lab grade
```

Cleanup

On **workstation**, run the **lab prepinternal-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab prepinternal-lab cleanup
```

This concludes the lab.

PREPARING TO DEPLOY AN INSTANCE

PERFORMANCE CHECKLIST

In this lab, you will create the resources required to launch an instance, including an image, a project network and subnetwork, and a flavor.

OUTCOMES

You should be able to:

- Create an image, and configure it as public.
- Create a project network and a subnetwork.
- Create a flavor.

Ensure that the **workstation** and overcloud virtual machines are started.

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

On **workstation**, run **lab prepinternal-lab setup**, which verifies that OpenStack services are running and the resources created in previous sections are available. The lab script creates the credentials files on **workstation** in **/home/student/architect1-production-rc** and **/home/student/operator1-production-rc**.

```
[student@workstation ~]$ lab prepinternal-lab setup
```

If you need administrative privileges, use the **architect1** account (password: **redhat**). The credentials file can be found on **workstation** as **/home/student/architect1-production-rc**.

1. Use the **http://materials.example.com/osp-web.qcow2** file to create an image named **rhel7-web** using the **operator1** user credentials. Configure this image to be **protected** and to use 10 GB as the minimum disk size, and 2048 MB as the minimum amount of RAM.
 - 1.1. From **workstation**, open a terminal. Download the **osp-web.qcow2** image file from the materials repository.

```
[student@workstation ~]$ wget http://materials.example.com/osp-web.qcow2
...output omitted...
2018-06-19 05:11:38 (211 MB/s) - 'osp-web.qcow2' saved [971767808/971767808]
```

- 1.2. Source the **operator1-production-rc** file to load the **operator1** user credentials.

```
[student@workstation ~]$ source ~/operator1-production-rc
```

- 1.3. Create an image named **rhel7-web** using the **osp-web.qcow2** image file previously downloaded. Configure this image to be **protected** and to use 10 GB as the minimum disk size, and 2048 MB as the minimum amount of RAM.

```
[student@workstation ~(operator1-production)]$ openstack image create \
--disk-format qcow2 \
--min-disk 10 \
--min-ram 2048 \
--protected \
--file osp-web.qcow2 \
rhel7-web
+-----+
| Field      | Value
+-----+
| checksum   | c37de831bdb9b50379bd2008c399275
| container_format | bare
| created_at | 2018-06-19T05:14:18Z
| disk_format | qcow2
| file       | /v2/images/a6b0ab1f-2ff8-4751-bed4-efcfb140d0fe/file
| id         | a6b0ab1f-2ff8-4751-bed4-efcfb140d0fe
| min_disk   | 10
| min_ram    | 2048
| name       | rhel7-web
| owner      | 00f7c7f68e7c4d72bab53dda8492ffcb
| properties  | direct_url='...', locations='...'
| protected   | True
| schema     | /v2/schemas/image
| size       | 971767808
| status     | active
| tags       |
| updated_at | 2018-06-19T05:14:33Z
| virtual_size | None
| visibility  | shared
+-----+
```

- 1.4. Verify that the **rhel7-web** image status is **active**.

```
[student@workstation ~(operator1-production)]$ openstack image list
+-----+-----+-----+
| ID           | Name      | Status |
+-----+-----+-----+
...output omitted...
| a6b0ab1f-2ff8-4751-bed4-efcfb140d0fe | rhel7-web | active |
...output omitted...
+-----+-----+-----+
```

2. Create a network named **production-network2** using the **operator1** user credentials, and a subnetwork on it named **production-subnet2**. Use the **192.168.2.0/24** range and the **172.25.250.254** DNS name server for the **production-subnet2** subnetwork.
- 2.1. Create a network named **production-network2**. By default, this network is created as an internal network.

```
[student@workstation ~(operator1-production)]$ openstack network create \
production-network2
+-----+
| Field      | Value
+-----+
| admin_state_up | UP
+-----+
```

availability_zone_hints	
availability_zones	
created_at	2018-06-19T05:27:09Z
description	
dns_domain	None
id	6fa4ac0d-409e-4bf3-8ff8-20680e817f53
ipv4_address_scope	None
ipv6_address_scope	None
is_default	False
is_vlan_transparent	None
mtu	1450
name	production-network2
project_id	00f7c7f68e7c4d72bab53dda8492ffcb
provider:network_type	None
provider:physical_network	None
provider:segmentation_id	None
qos_policy_id	None
revision_number	3
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	
tags	
updated_at	2018-06-19T05:27:09Z

- 2.2. Create a subnetwork named **production-subnet2** on the **production-network2** network. Use the **192.168.2.0/24** range and the **172.25.250.254** DNS name server for that subnetwork.

[student@workstation ~](operator1-production)]\$ openstack subnet create \
production-subnet2 \
--network production-network2 \
--dns-nameserver 172.25.250.254 \
--subnet-range 192.168.2.0/24
+-----+-----+-----+
Field Value
+-----+-----+-----+
allocation_pools 192.168.2.2-192.168.2.254
cidr 192.168.2.0/24
created_at 2018-06-19T05:33:24Z
description
dns_nameservers 172.25.250.254
enable_dhcp True
gateway_ip 192.168.2.1
host_routes
id 725d62c8-d99d-44f6-aa7b-c265f4f330eb
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name production-subnet2
network_id 6fa4ac0d-409e-4bf3-8ff8-20680e817f53
project_id 00f7c7f68e7c4d72bab53dda8492ffcb
revision_number 0
segment_id None

service_types		
subnetpool_id	None	
tags		
updated_at	2018-06-19T05:33:24Z	

3. Create a flavor named **default-extra-disk**. Configure this flavor with a 10 GB root disk, 2048 MB of RAM, two VCPUs, a 5 GB ephemeral disk, a 1024 MB swap disk, and use 42 as the ID of the flavor.

- 3.1. Since creating a flavor must be done with an administrative account, source the **architect1-production-rc** file to load the **architect1** user credentials.

```
[student@workstation ~ (operator1-production)]$ source ~/architect1-production-rc
[student@workstation ~ (architect1-production)]$
```

- 3.2. Create a flavor named **default-extra-disk** with a 10 GB root disk, 2048 MB of RAM, two VCPUs, a 5 GB ephemeral disk, a 1024 MB swap disk, and use 42 as the ID of the flavor.

Field	Value	
OS-FLV-DISABLED:disabled	False	
OS-FLV-EXT-DATA:ephemeral	5	
disk	10	
id	42	
name	default-extra-disk	
os-flavor-access:is_public	True	
properties		
ram	2048	
rxtx_factor	1.0	
swap	1024	
vcpus	2	

4. Configure the **rhel7-web** image to be **public**.

- 4.1. This must be done as an administrative user, so verify you have the **architect1** credentials. Modify the **rhel7-web** image visibility to be **public**.

```
[student@workstation ~ (architect1-production)]$ openstack image set \
--public \
```

4.2. Verify that the **rhel7-web** image visibility is now **public**.

```
[student@workstation ~](architect1-production)]$ openstack image show rhel7-web
+-----+-----+
| Field      | Value
+-----+-----+
| checksum   | c37de831bdbc9b50379bd2008c399275
| container_format | bare
| created_at | 2018-06-19T05:14:18Z
| disk_format | qcow2
| file       | /v2/images/a6b0ab1f-2ff8-4751-bed4-efcfb140d0fe/file
| id         | a6b0ab1f-2ff8-4751-bed4-efcfb140d0fe
| min_disk   | 10
| min_ram   | 2048
| name       | rhel7-web
| owner      | 00f7c7f68e7c4d72bab53dda8492ffcb
| properties | direct_url='...', locations='...'
| protected  | True
| schema     | /v2/schemas/image
| size       | 971767808
| status     | active
| tags       |
| updated_at | 2018-06-19T05:45:11Z
| virtual_size | None
| visibility | public
+-----+-----+
```

Evaluation

On **workstation**, run the **lab_prepinternal-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab_prepinternal-lab grade
```

Cleanup

On **workstation**, run the **lab_prepinternal-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab_prepinternal-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Instance virtual disks are based on templates known as images.
- These images are managed by the Image service, and can be enlarged at instance deployment time to adjust themselves to the size specified in the flavor.
- Limits can be set on an image to define the minimum disk size and RAM required to launch an instance with the image.
- The defined minimum disk size and RAM of an image should be accommodated by the flavor chosen to launch an instance.
- Public images are accessible by all projects, but they can be made private to limit access to the project from which the image was uploaded.
- A flavor defines the resources to be used by an instance, including the number of VCPUs, memory, or disk size.
- Flavors can only be created by users with administrative privileges.
- Network connectivity is provided to instances by the OpenStack Networking Service, which supports different networking elements such as networks, subnetworks, and routers.
- IP addressing is supported by subnetworks. Subnetworks on different networks can use the same IP addressing thanks to network isolation.
- The kernel provided by Red Hat Enterprise Linux 7 includes network namespaces, which cover network access with a layer of abstraction by virtualizing access to the network resources such as ports and interfaces.

CHAPTER 6

DEPLOYING AN INSTANCE

GOAL

Launch and verify an instance.

OBJECTIVES

- Launch an instance to illustrate the minimum OpenStack resource preparation required.
- Verify the functionality of an instance.

SECTIONS

- Launching an Instance (and Guided Exercise)
- Verifying the Functionality of an Instance (and Guided Exercise)

LAB

- Deploying an Instance

LAUNCHING AN INSTANCE

OBJECTIVE

After completing this section, students should be able to launch an instance to illustrate the minimum OpenStack resource preparation required.

LAUNCHING AN INSTANCE

The first chapter covered how to launch an instance using the dashboard and the unified CLI. The minimum resources needed to launch a new instance are a flavor, an image, and a network. Using the CLI and the dashboard, you viewed available flavors and images, and launched an instance. You viewed the console logs connected to a running instance console.



NOTE

If there is only one network defined in a project, that network will be used automatically when launching an instance.

VERIFYING AVAILABLE RESOURCES

You can use the **openstack** command to list and examine the available resources in Red Hat OpenStack Platform. The **openstack** command can list available flavors, images, instances, and networks. An administrative user can also use this command to list available projects, users, and roles. You can use the **openstack show** command to view the details of a chosen resource. You can use the information provided by this command to help launch an instance or perform changes to the Red Hat OpenStack Platform environment.

Network Verification

When you are in a project with multiple resources of the same type, for example networks, the specific resource to be used must be specified when launching an instance. Unless you specify one of the required multiple resources, the instance will not be created.

```
[user@demo ~(user-demo)]$ openstack server create --flavor default \
--image rhel7 demo-server2
Multiple possible networks found, use a Network ID to be more specific. (HTTP 409)
(Request-ID: req-841a675f-c762-445d-9049-8cb5e6c6649d)
```

Listing the available resources can help identify the correct one to use. Considering the previous example, listing the networks would help find the one needed to launch the instance. The output of the following command indicates that two usable networks exist in the project.

```
[user@demo ~(user-demo)]$ openstack network list
+-----+-----+-----+
| ID          | Name        | Subnets           |
+-----+-----+-----+
| a1273a7b-2a73-42ef-a97c- | demo-network1 | 38e05db4-a397-491a-a8f7- |
| 4684aef83f96   |             | 8b3220799aa2     |
| c41edb85-b344-4bf1-abd6- | demo-network2 | 27275a69-e720-4c3c-9529- |
| a842000dba01    |             | 9061bb5af0af     |
```

To launch an instance, the correct network needs to be identified and used with the **openstack server create** command. This lets OpenStack know which resource that you want to associate with a particular instance.

```
[user@demo ~(user-demo)]$ openstack server create --flavor default \
--image rhe17 --nic net-id=demo-network2 demo-server2
```

Deleting Instances

The **openstack server delete** command deletes OpenStack instances. The name or ID of the instance must be specified when using this command.

```
[user@demo ~(user-demo)]$ openstack server delete demo-server2
```

It is a good practice to list available resources, after every change made, to confirm the intended change was made. In this example, you should list available instances to verify that the proper instance was deleted.

```
[user@demo ~(user-demo)]$ openstack server list
```

Verifying Available Resources and Launching an Instance

The following steps outline the process of listing available resources and launching an instance using the dashboard.

1. Log in to the dashboard as an administrative user. View the available flavors by navigating to Admin → Compute → Flavors.
2. Log in to the dashboard as a normal user. View the available images by navigating to Project → Compute → Images.
3. View the available networks by navigating to Project → Network → Networks.
4. Using the available resources from the previous steps, launch a new instance.
5. After a few seconds, verify the status of the instance.

Verifying Available Resources and Launching an Instance from the CLI

The following steps outline the process of listing available resources and launching an instance using the CLI.

1. Source the appropriate identity environment file.
2. List the available images with the **openstack image list** command.
3. Use the **openstack flavor list** command to display the list of available flavors. If needed, view the details of a flavor with the **openstack flavor show** command.
4. List the available networks with the **openstack network list** command.
5. Using the resources from the previous steps, launch a new instance with the **openstack server create** command.
6. List the current instances with the **openstack server list** command. Confirm the status of the new instance.



REFERENCES

Further information is available in the Virtual Machine Instances section of the *Instances and Images Guide* for Red Hat OpenStack Platform at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/instances_and_images_guide/

LAUNCHING AN INSTANCE

In this guided exercise you will launch an instance with the dashboard. You will also delete that instance and launch a new instance using the CLI.

OUTCOMES

You should be able to:

- Log in to the dashboard and launch an instance using existing resources.
- Delete an instance using the CLI.
- Create a new instance from the CLI using existing resources.

Confirm the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab depinternal-launch** script with the **setup** argument. It creates the required components for this exercise, such as users, passwords, images, and networks.

```
[student@workstation ~]$ lab depinternal-launch setup
```

- ▶ 1. On **workstation**, open Firefox and browse to **http://dashboard.overcloud.example.com**.
- ▶ 2. Log in to the dashboard using **developer1** as the user name and **redhat** as the password.
- ▶ 3. Click the Project menu in the top right corner of the window to verify that **finance** is the current project.
- ▶ 4. Launch an instance named **finance-server2** using the **default** flavor, the **rhel7** image, and the **finance-network1** network.
 - 4.1. Click the Instances subtab under the Compute tab.
 - 4.2. Click Launch Instance.
 - 4.3. In the Details tab, enter **finance-server2** as the Instance Name.
 - 4.4. On the Source tab, click Select Boot Source → Image.
 - 4.5. On the Source tab, click the up arrow button on the same row as the image to allocate the **rhel7** image. Click No under Create New Volume.
 - 4.6. On the Flavor tab, click the up arrow button on the same row as the flavor to allocate the **default** flavor.
 - 4.7. On the Networks tab, if the **finance-network1** network is not already allocated, click the up arrow button on the same row as the network to allocate the **finance-network1** network.
 - 4.8. Click Launch Instance to launch **finance-server2**.

- ▶ 5. Wait a few seconds and then verify the status of the instance in the Power State column. The instance should be in the **Running** state. Once the instance is running, sign out of the dashboard.
- ▶ 6. On **workstation**, open a terminal and source the **developer1** user's identity environment file, **/home/student/developer1-finance-rc**.

```
[student@workstation ~]$ source ~/developer1-finance-rc  
[student@workstation ~(developer1-finance)]$
```

- ▶ 7. Delete the instance created in the dashboard. Verify the instance is deleted.
 - 7.1. List all the available instances.

```
[student@workstation ~(developer1-finance)]$ openstack server list -f json  
[  
 {  
   "Status": "ACTIVE",  
   "Name": "finance-server2",  
   "Image": "rhel7",  
   "ID": "bc8560ae-d9fa-4fdd-847d-fa58f8fd64ac",  
   "Flavor": "default",  
   "Networks": "finance-network1=192.168.1.N"  
 }  
 ]
```

- 7.2. Delete the existing instance named **finance-server2**.

```
[student@workstation ~(developer1-finance)]$ openstack server delete \  
finance-server2
```

- 7.3. List the instances to verify the instance was deleted.

```
[student@workstation ~(developer1-finance)]$ openstack server list
```

- ▶ 8. Launch a new instance named **finance-server3** using existing resources. Verify the instance is running.
 - 8.1. Create a new instance called **finance-server3**. Use the **rhel7** image, the **default** flavor, and the **finance-network1** network.

```
[student@workstation ~(developer1-finance)]$ openstack server create \  
--image rhel7 \  
--flavor default \  
--nic net-id=finance-network1 \  
--wait finance-server3  
+-----+  
| Field | Value |  
+-----+  
| OS-DCF:diskConfig | MANUAL |  
| OS-EXT-AZ:availability_zone | nova |  
| OS-EXT-STS:power_state | Running |  
| OS-EXT-STS:task_state | None |  
| OS-EXT-STS:vm_state | active |
```

OS-SRV-USG:launched_at	2018-06-12T06:12:28.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	finance-network1=192.168.1.P
adminPass	<i>PcZUCE2wL6ss</i>
config_drive	
created	2018-06-12T06:12:13Z
flavor	default (5ed66a7a-7e62-46ad-9ab7-1cc42a18fc0c)
hostId	c624...44a0
id	3172b88d-7259-44e4-9016-25fb7d4f5bde
image	rhel7 (d6a496f1-87b2-4176-bd61-1700ce3f6887)
key_name	None
name	finance-server3
progress	0
project_id	438f276649894dcdbedb36ba1dc5296f
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-12T06:12:29Z
user_id	17dd9d37aaff43b7beb31771a6e1227c
volumes_attached	

8.2. List all the available instances to verify the **finance-server3** instance is running.

```
[student@workstation ~](developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server3",
    "Image": "rhel7",
    "ID": "e2304849-8e71-4702-97e1-5cada99d83bf",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.P"
  }
]
```

Cleanup

On **workstation**, run the **lab depinternal-launch cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab depinternal-launch cleanup
```

This concludes the guided exercise.

VERIFYING THE FUNCTIONALITY OF AN INSTANCE

OBJECTIVE

After completing this section, students should be able to verify the functionality of an instance.

VERIFYING A RUNNING INSTANCE

Red Hat OpenStack Platform provides many ways to verify running instances. This can help troubleshoot potential problems or explore current limitations. Verification can be performed using the dashboard or from the unified CLI.

For example, developers using instances could report that they do not have the correct amount of resources available. Perhaps the concern is over the amount of available RAM, CPUs, or disk space. In this situation, there is a need to verify the configuration of the instances. This task can be performed from Red Hat OpenStack Platform or from the operating system of the instances. The Red Hat OpenStack Platform values can be compared with the corresponding resource values presented by the operating system.

Verifying Dedicated Resources

The following commands show how to take a closer look at OpenStack resources and compare the gathered information with the values seen from inside of a running instance. First confirm the instance is in the **Running** state and confirm the image, network, and flavor used for creating that instance are the correct ones. To verify this, use the **openstack server show** command.

```
[user@demo ~(user-demo)]$ openstack server show demo-server1
+-----+
| Field           | Value
+-----+
| OS-DCF:diskConfig | MANUAL
|
| OS-EXT-AZ:availability_zone | nova
|
| OS-EXT-STS:power_state | Running
|
| OS-EXT-STS:task_state | None
|
| OS-EXT-STS:vm_state | active
|
| OS-SRV-USG:launched_at | 2018-06-12T15:46:34.000000
|
| OS-SRV-USG:terminated_at | None
|
| accessIPv4 |
|
| accessIPv6 |
|
| addresses | demo-network1=192.168.1.7
|
```

```

| config_drive          | 
| 
| created               | 2018-06-12T15:46:19Z
| 
| flavor                | default (db97c88d-36a3-4ecc-b66f-7d99b59f1fbc)
| 
| hostId               | 2e30...efb8
| 
| id                    | 81519aa2-b3af-4a05-9e0d-81b1660a16b4
| 
| image                 | rhel7-web (2cb5074b-aa91-4240-9d97-a4b5c0079853)
| 
| key_name              | None
| 
| name                  | demo-server1
| 
| progress               | 0
| 
| project_id            | 1bfc1e465c0442aab6d5fa387ae1c244
| 
| properties             | 
| 
| security_groups        | name='default'
| 
| status                 | ACTIVE
| 
| updated                | 2018-06-13T08:27:09Z
| 
| user_id                | c0a85a6918b14200884a276d92415872
| 
| volumes_attached       | 
| 
+-----+
+-----+

```

The previous command output confirms the status of the instance and it includes the image, flavor, and network that are used by the instance. Knowing which image was used to create an instance allows you to verify the image by using the **openstack image show** command. The output of this command displays the name, disk format, file name, ID, status, and size of the image. This information allows the user to find and access the image file itself.

```

[user@demo ~] $ openstack image show rhel7-web
+-----+-----+
| Field      | Value
+-----+-----+
| checksum   | d53be3734ab7e0769f8ece711df76250
| container_format | bare
| created_at | 2018-06-12T14:12:23Z
| disk_format | qcow2
| file       | /v2/images/2cb5074b-aa91-4240-9d97-a4b5c0079853/file
| id         | 2cb5074b-aa91-4240-9d97-a4b5c0079853
| min_disk   | 10
| min_ram    | 2048
| name       | rhel7-web
| owner      | 1bfc1e465c0442aab6d5fa387ae1c244
| properties | direct_url='rbd://e60f...01fa'
| 
```

protected	True
schema	/v2/schemas/image
size	967835648
status	active
tags	
updated_at	2018-06-12T14:12:35Z
virtual_size	None
visibility	public

You can also verify the hardware limitations specified in the flavor used by the instance. The **openstack flavor show** command displays this information. This command provides information about the amount of RAM, the number of CPUs, and the disk sizes.

[user@demo ~(user-demo)]\$ openstack flavor show default		
Field	Value	
OS-FLV-DISABLED:disabled	False	
OS-FLV-EXT-DATA:ephemeral	0	
access_project_ids	None	
disk	10	
id	db97c88d-36a3-4ecc-b66f-7d99b59f1fbc	
name	default	
os-flavor-access:is_public	True	
properties		
ram	2048	
rxtx_factor	1.0	
swap		
vcpus	2	

You can also get detailed information about the network used by the instance.

[user@demo ~(user-demo)]\$ openstack network show demo-network1		
Field	Value	
admin_state_up	UP	
availability_zone_hints		
availability_zones	nova	
created_at	2018-06-12T09:01:11Z	
description		
dns_domain	None	
id	49f4ab91-42f9-4944-8694-81ba503991b3	
ipv4_address_scope	None	
ipv6_address_scope	None	
is_default	None	
is_vlan_transparent	None	
mtu	1450	
name	demo-network1	
port_security_enabled	True	
project_id	1bfc1e465c0442aab6d5fa387ae1c244	
provider:network_type	None	
provider:physical_network	None	

provider:segmentation_id	None
qos_policy_id	None
revision_number	4
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	dbdb156f-8c2b-4ffa-82ff-d6b4976aed2a
tags	
updated_at	2018-06-12T09:01:29Z

The information displayed by this command, especially the name or ID of the network and subnet, is useful for viewing the instance's network information. The subnet configuration holds the IP allocation pools and the gateway IP address. You can list all the available subnets in a project by using the **openstack subnet list** command and you can access the details of a specific subnet with the **openstack subnet show** command.

For example, the following command displays the details of the subnet which ID has been retrieved by the previous **openstack network show demo-network1** command. The output shows the allocated IP range, the CIDR address of the network, the gateway IP address, and other information.

[user@demo ~(user-demo)]\$ openstack subnet show dbdb156f-8c2b-4ffa-82ff-d6b4976aed2a	
Field	Value
allocation_pools	192.168.1.2-192.168.1.254
cidr	192.168.1.0/24
created_at	2018-06-12T09:01:29Z
description	
dns_nameservers	172.25.250.254
enable_dhcp	True
gateway_ip	192.168.1.1
host_routes	
id	dbdb156f-8c2b-4ffa-82ff-d6b4976aed2a
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	demo-subnet1
network_id	49f4ab91-42f9-4944-8694-81ba503991b3
project_id	1bfc1e465c0442aab6d5fa387ae1c244
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-06-12T09:01:29Z

You can also use the subnet name as an argument to the **openstack subnet show** command instead of the subnet ID.

Any errors that are discovered in the subnet configuration can be fixed using the **openstack subnet set** command. The following example updates the range of the DHCP allocation pool.

```
[user@demo ~(user-demo)]$ openstack subnet set \
--allocation-pool start=192.168.1.4,end=192.168.1.50 demo-subnet1
```

You can use the **openstack help subnet set** command to list all the available options.

With all the information from the previous command outputs, you can log in to the instance and run commands from the operating system command line and verify the information. You can review the disks, CPU, and RAM specifications by using the **df**, **fdisk**, **lscpu**, and **free** commands. Comparing the previously gathered information with the actual configuration from the operating system CLI, you can verify and correct the configuration of that instance. Viewing the resources used to create an instance and information about those resources can be valuable when you investigate failures.

PAUSING AND RESUMING A RUNNING INSTANCE

One reason for using Red Hat OpenStack Platform is its elasticity, for example its *pause* functionality. OpenStack is capable of temporarily stopping and restarting a running instance. For example, if an instance is no longer needed, but the physical resources of the hypervisor are needed for other tasks, use the **openstack server pause** command to free some of the physical resources of the hypervisor. This command pauses an instance by storing its state in the memory of the hypervisor. A paused instance continues to run in a frozen state, but from the outside the instance appears stopped or crashed.

Instead of terminating instances, the **pause** command can be used when servers cannot be restarted. For example, to back up an important instance that is writing data to disks, the instance can be paused and the backup performed while the instance is in the paused state.

```
[user@demo ~(user-demo)]$ openstack server pause demo-server1
```

After the backup is finished, the instance can be resumed to run normally. This is accomplished with the **openstack unpause** command.

```
[user@demo ~(user-demo)]$ openstack server unpause demo-server1
```

Managing an Instance

The following steps outline the process for starting, stopping, pausing, and resuming a instance.

1. Source the appropriate identity environment file.
2. List the available instances with the **openstack server list** command.
3. View the details of the running instance with the **openstack server show** command.
4. Stop the running instance with the **openstack server stop** command.
5. Start the instance with the **openstack server start** command.
6. Pause a running instance with the **openstack server pause** command.
7. Resume the instance with the **openstack server unpause** command.
8. Retrieve the URL of the instance console with the **openstack console url show** command and use a web browser to access the console.
9. Ping various available IP addresses, for example the DHCP server or gateway. Verify connectivity to the external network.

TROUBLESHOOTING AN INSTANCE

Sometimes troubleshooting an instance requires that you get more detailed information about what happened to an instance, for example if an instance cannot be launched.

Troubleshooting Launch Errors in the Dashboard

When using the dashboard to launch instances, if the operation fails, a generic ERROR message is displayed. Determining the actual cause of the failure may require the use of the command-line tools.

Use the `openstack server list` command to locate the name or ID of the instance, and then use this as an argument to the `openstack server show` command. One of the items returned is the error condition. The most common value returned is **NoValidHost**. This error indicates that no valid host with enough available resources was found. Because of the absence of available resources, the compute service could not launch a new instance. To work around this issue, investigate the cause of the error in the log files. You may need to choose a smaller instance size or increase the overcommit allowances for the environment.



NOTE

To host a given instance, the compute node must have enough CPUs, RAM, and disk space for the storage associated with the instance.

Additional information about the instance issues may be found in one of the following log files:

Compute Service Log Files

SERVICE	CONTAINER	NODE	LOG PATH
Compute API service	nova_api	controller	/var/log/containers/nova/nova-api.log and /var/log/containers/httpd/nova-api/*
Compute service	nova_compute	compute	/var/log/containers/nova/nova-compute.log
Compute Conductor service	nova_conductor	controller	/var/log/containers/nova/nova-conductor.log
Compute VNC console authenticator server	nova_consoleauth	controller	/var/log/containers/nova/nova-consoleauth.log
Compute NoVNC Proxy service	nova_vnc_proxy	controller	/var/log/containers/nova/nova-novncproxy.log
Compute Scheduler service	nova_scheduler	controller	/var/log/containers/nova/nova-scheduler.log

Service	Container	Node	Log Path
Compute Metadata service	nova_metadata	controller	/var/log/containers/nova/nova-api-metadata.log

Start with the **/var/log/containers/nova/nova-conductor.log** and the **/var/log/containers/nova/nova-scheduler.log**. They often have valuable information in troubleshooting an instance error. You may need to expand your search from there as you investigate instance issues.

Troubleshooting Networking Issues

Additional commands and procedures can be used to troubleshoot Red Hat OpenStack Platform networking service issues. Debugging networking devices can be performed by using the **ip** or **ovs-vsctl show** commands, together with the **ovs-dpctl show** command. The **ip** command displays all the physical and virtual devices.

```
[user@demo ~(user-demo)]$ ip addr show
...output omitted...
3: eth1: BROADCAST,MULTICAST,UP,LOWER_UP mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 52:54:00:01:fa:0b brd ff:ff:ff:ff:ff:ff
    inet 172.24.250.11/24 brd 172.24.250.255 scope global eth1
        valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe01:fa0b/64 scope link
        valid_lft forever preferred_lft forever
4: ovs-system: BROADCAST,MULTICAST mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether fa:99:09:63:89:a3 brd ff:ff:ff:ff:ff:ff
5: br-ex: BROADCAST,MULTICAST,UP,LOWER_UP mtu 1500 qdisc noqueue state UNKNOWN
qlen 1000
    link/ether 22:b3:01:d9:81:49 brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.11/24 brd 172.25.250.255 scope global br-ex
        valid_lft forever preferred_lft forever
    inet6 fe80::20b3:1fff:fed9:8149/64 scope link
        valid_lft forever preferred_lft forever
6: br-int: BROADCAST,MULTICAST mtu 1450 qdisc noop state DOWN qlen 1000
    link/ether 7a:96:75:ef:85:47 brd ff:ff:ff:ff:ff:ff
7: br-tun: BROADCAST,MULTICAST mtu 1500 qdisc noop state DOWN qlen 1000
    link/ether 32:91:f9:06:35:4b brd ff:ff:ff:ff:ff:ff
9: qbr669fbfc8-46: BROADCAST,MULTICAST,UP,LOWER_UP mtu 1450 qdisc noqueue state UP
qlen 1000
    link/ether 82:06:65:22:33:5c brd ff:ff:ff:ff:ff:ff
10: qvo669fbfc8-46@qvb669fbfc8-46: BROADCAST,MULTICAST,PROMISC,UP,LOWER_UP mtu
1450 qdisc noqueue master ovs-system state UP qlen 1000
    link/ether 36:64:8a:42:f7:03 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::3464:8aff:fe42:f703/64 scope link
        valid_lft forever preferred_lft forever
...output omitted...
```

The **ovs-vsctl show** command displays the interfaces and bridges in a virtual switch.

```
[root@demo ~]# ovs-vsctl show
...output omitted...
Bridge br-int
```

```
Controller "tcp:127.0.0.1:6633"
    is_connected: true
fail_mode: secure
Port patch-tun
    Interface patch-tun
        type: patch
        options: {peer=patch-int}
Port "tap8f2d7666-fa"
    tag: 1
    Interface "tap8f2d7666-fa"
        type: internal
Port int-br-ex
    Interface int-br-ex
        type: patch
        options: {peer=phy-br-ex}
Port "qvoc1669e2f-2c"
    tag: 1
    Interface "qvoc1669e2f-2c"
Port "qvoe84ac574-a7"
    tag: 1
    Interface "qvoe84ac574-a7"
Port "qvo669fbfc8-46"
    tag: 1
    Interface "qvo669fbfc8-46"
Port "qvo6749ecd5-4d"
    tag: 1
    Interface "qvo6749ecd5-4d"
...output omitted...
```

The **ovs-dpctl show** command displays the data paths on a switch.

```
[root@demo ~]# ovs-dpctl show
system@ovs-system:
    lookups: hit:46746 missed:290 lost:0
    flows: 2
    masks: hit:52369 total:1 hit/pkt:1.11
    port 0: ovs-system (internal)
    port 1: eth0
    port 2: br-ex (internal)
    port 3: br-int (internal)
    port 4: br-tun (internal)
    port 5: tap8f2d7666-fa (internal)
    port 6: qvo669fbfc8-46
    port 7: qvo6749ecd5-4d
    port 8: qvoc1669e2f-2c
    port 9: qvoe84ac574-a7
```

To correctly identify which network is connected to an instance, it is a good practice to use the **openstack network list** command. The output of this command shows the names and IDs of the available networks. Knowing the network ID helps to access the correct interfaces and bridges on the controller node.

```
[user@demo ~(user-demo)]$ openstack network list
+-----+-----+
+-----+-----+
```

ID	Name	Subnets
49f4ab91-42f9-4944-8694-81ba503991b3 demo-network1 dbdb...ed2a		
0177f4ee-87d1-47e0-b0e1-c42171311dce demo-network2 ec5c...3fca		

The available network namespaces can be listed with the **ip netns list** command on the controller node.

```
[root@demo ~]# ip netns list
qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 (id: 0)
qdhcp-0177f4ee-87d1-47e0-b0e1-c42171311dce (id: 1)
```

Notice how the network namespace incorporates the ID of the actual OpenStack networks. This helps to access the correct interfaces.

After identifying the correct network interface, use that interface ID with other commands to troubleshoot networking issues. The **ip netns exec** command can be used to execute network commands within a network namespace. For example, the **ping** command can be used to test connections.

```
[root@demo ~]# ip netns exec qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 \
ping -c 3 192.168.1.7
PING 192.168.1.7 (192.168.1.7) 56(84) bytes of data.
64 bytes from 192.168.1.7: icmp_seq=1 ttl=64 time=7.75 ms
64 bytes from 192.168.1.7: icmp_seq=2 ttl=64 time=2.49 ms
64 bytes from 192.168.1.7: icmp_seq=3 ttl=64 time=2.14 ms

--- 192.168.1.7 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2004ms
rtt min/avg/max/mdev = 2.141/4.130/7.753/2.565 ms
```

Exploring OpenStack Networking

Several commands can be used to explore the OpenStack network configuration. Previous examples showed how to execute commands within a network namespace. This technique can be used to execute any command that runs within the network namespace. For example, the **ip addr show** command displays the DHCP server IP address:

```
[root@demo ~]# ip netns exec qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 ip addr
show
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 65536 qdisc noqueue state UNKNOWN group default
    qlen 1000
        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
19: tap8f2d7666-fa: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state
UNKNOWN group default qlen 1000
```

```
link/ether fa:16:3e:6d:3d:15 brd ff:ff:ff:ff:ff:ff
inet 192.168.1.2/24 brd 192.168.1.255 scope global tap8f2d7666-fa
    valid_lft forever preferred_lft forever
inet 169.254.169.254/16 brd 169.254.255.255 scope global tap8f2d7666-fa
    valid_lft forever preferred_lft forever
inet6 fe80::f816:3eff:fe6d:3d15/64 scope link
    valid_lft forever preferred_lft forever
```

The previous output indicates that the DHCP server is running on the **tap8f2d7666-fa** interface and has an IP address of **192.168.1.2**. The DHCP interface connections on the internal Red Hat OpenStack Platform switch can be displayed using **ovs-vsctl show**.

```
[root@demo ~]# ovs-vsctl show
...output omitted...
Bridge br-int
    Controller "tcp:127.0.0.1:6633"
        is_connected: true
    fail_mode: secure
    Port patch-tun
        Interface patch-tun
            type: patch
            options: {peer=patch-int}
    Port "tapfcfd4fe60-06"
        tag: 1
        Interface "tapfcfd4fe60-06"
            type: internal
    Port "tap8f2d7666-fa"
        tag: 2
        Interface "tap8f2d7666-fa"
            type: internal
    Port int-br-ex
        Interface int-br-ex
            type: patch
            options: {peer=phy-br-ex}
    Port br-int
        Interface br-int
            type: internal
...output omitted...
```

The previous output shows that the switch uses the **tap8f2d7666-fa** interface as an internal port of the **br-int** bridge.

Using all of this information can help to explore network issues. It can be used to check if a connection between different instances is possible, for example, to check if a connection from the DHCP server to the web server running on one of the instances is possible. This can be tested by establishing a connection between the DHCP server and the TCP port of a service running on one of the instances. The following example uses the **nc** command to make a connection from the DHCP server to the TCP port 80 on the running instance (192.168.1.7).

```
[root@demo ~]# ip netns qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 nc 192.168.1.7
80
```

Depending on the setup, the possible outcomes for the previous command are listed below:

- The port is open and accessible, and the connection can be established.

- The instance cannot be reached, and a **No route to host** warning is displayed. This happens when the instance has the wrong network configuration.
- The connection cannot be established. Possible causes are: the service did not start; the firewall inside the instance blocks that port; or the security group in Red Hat OpenStack Platform does not allow a connection using that port.



REFERENCES

Further information is available in the Troubleshooting section of the *Logging, Monitoring, and Troubleshooting Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/logging_monitoring_and_troubleshooting_guide/

VERIFYING THE FUNCTIONALITY OF AN INSTANCE

In this guided exercise you will connect to a running instance and verify the flavor settings. You will also pause and stop an instance using the CLI.

OUTCOMES

You should be able to:

- Open a console and log in to a running instance.
- Verify the flavor settings of a running instance.
- Stop, pause, and delete a running instance using the CLI.

Ensure that **workstation** and the overcloud's virtual machines are started.

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

On **workstation**, run **lab depinternal-verify setup**, which verifies that the OpenStack services are running and the resources created in previous sections are available.

```
[student@workstation ~]$ lab depinternal-verify setup
```

- ▶ 1. On **workstation**, open a terminal and source the **developer1** user's identity environment file.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~ (developer1-finance)]$
```

- ▶ 2. Find the details of the flavor used by the running instance.

2.1. List all the available instances to find the name of the running instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server3",
    "Image": "rhel7",
    "ID": "deca6e99-f662-4109-811e-5141c0406564",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.N"
  }
]
```

2.2. Access the instance's details to retrieve its flavor.

```
[student@workstation ~(developer1-finance)]$ openstack server show \
finance-server3
+-----+-----+
| Field | Value |
+-----+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2018-06-12T06:32:56.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4 | |
| accessIPv6 | |
| addresses | finance-network1=192.168.1.N |
| config_drive | |
| created | 2018-06-12T06:32:40Z |
| flavor | default (5ed66a7a-7e62-46ad-9ab7-1cc42a18fc0c) |
| hostId | c624...44a0 |
| id | deca6e99-f662-4109-811e-5141c0406564 |
| image | rhel7 (d6a496f1-87b2-4176-bd61-1700ce3f6887) |
| key_name | None |
| name | finance-server3 |
| progress | 0 |
| project_id | 438f276649894dcdbedb36ba1dc5296f |
| properties | |
| security_groups | name='default' |
| status | ACTIVE |
| updated | 2018-06-12T06:32:57Z |
| user_id | 17dd9d37aaff43b7beb31771a6e1227c |
| volumes_attached | |
+-----+-----+
```

2.3. Review the specifications of the **default** flavor.

```
[student@workstation ~(developer1-finance)]$ openstack flavor show default
+-----+-----+
| Field | Value |
+-----+-----+
| OS-FLV-DISABLED:disabled | False |
| OS-FLV-EXT-DATA:ephemeral | 0 |
| access_project_ids | None |
| disk | 10 |
| id | 5ed66a7a-7e62-46ad-9ab7-1cc42a18fc0c |
| name | default |
| os-flavor-access:is_public | True |
| properties | |
| ram | 2048 |
| rxtx_factor | 1.0 |
| swap | |
| vcpus | 2 |
+-----+-----+
```

► 3. Log in to the instance.

- 3.1. Retrieve the URL for the console connection.

```
[student@workstation ~(developer1-finance)]$ openstack console url show \
finance-server3
+-----+
| Field | Value |
+-----+
| type  | novnc |
| url   | http://172.25.250.50:6080/vnc_auto.html?token=2f25...438f |
+-----+
```

- 3.2. On **workstation**, open Firefox and browse to the URL from the previous output, **http://172.25.250.50:6080/vnc_auto.html?token=2f252beb-4fe3-4e97-bf85-073533b1438f**.

- 3.3. From the console in Firefox, log in to the **finance-server3** instance as **root**. The password is **redhat**.

```
Red Hat Enterprise Linux Server release 7.5 (Maipo)
Kernel 3.10.0-862.el7.x86_64 on an x86_64

small image
finance-server3 login: root
Password: redhat
[root@finance-server3 ~]#
```

► 4. Review the instance RAM, disk, and CPU configuration.

- 4.1. Retrieve the value of the **MemTotal** variable from the **/proc/meminfo** special file to ensure that the RAM amount matches the one defined by the flavor, that is, 2048 MiB.

```
[root@finance-server3 ~]# grep MemTotal /proc/meminfo
MemTotal:      1882616 kB
```



NOTE

The kernel reserves some memory at boot time for *kdump*. *kdump* is a kernel feature that saves a copy of the system memory to disk for later analysis in the event of a kernel crash. This explains the missing 200 MiB in the previous output, 2097152 MiB - 1882616 MiB.

- 4.2. Use the **df** command to ensure that the instance has a 10 GB disk, as defined by the flavor.

```
[root@finance-server3 ~]# df -h
Filesystem      Size  Used Avail Use% Mounted on
/dev/vda1        10G   1.1G  9.0G  11% /
devtmpfs        898M     0  898M   0% /dev
tmpfs          920M     0  920M   0% /dev/shm
tmpfs          920M  8.5M  911M   1% /run
tmpfs          920M     0  920M   0% /sys/fs/cgroup
```

```
tmpfs          184M      0   184M  0% /run/user/0
```

- 4.3. Determine the number of CPUs that the instance is using. Ensure that this number matches the VCPUs number defined by the flavor.

```
[root@finance-server3 ~]# lscpu
Architecture:           x86_64
CPU op-mode(s):         32-bit, 64-bit
Byte Order:             Little Endian
CPU(s):                 2
On-line CPU(s) list:   0,1
Thread(s) per core:    1
Core(s) per socket:    1
Socket(s):              2
NUMA node(s):           1
Vendor ID:              GenuineIntel
CPU family:             6
Model:                  13
Model name:             QEMU Virtual CPU version 2.5+
Stepping:               11
CPU MHz:                2294.872
BogoMIPS:               4589.74
Hypervisor vendor:     KVM
Virtualization type:   full
L1d cache:              32K
L1i cache:              32K
L2 cache:                4096K
L3 cache:                16384K
NUMA node0 CPU(s):      0,1
Flags:                  fpu vme de pse tsc msr pae mce cx8 apic sep mtrr pge mca
                        cmov pat pse36 clflush mmx fxsr sse sse2 ss syscall nx lm constant_tsc rep_good
                        nopl dni ssse3 cx16 x2apic hypervisor lahf_lm
```

- 5. Verify the networking for the instance then pause, unpause, and stop the instance to see how it reacts.

- 5.1. Use the **ping** command from the instance to reach the DHCP server defined for the network. Leave the **ping** command running as it will be used in the following steps.

```
[root@finance-server3 ~]# ping 192.168.1.2
PING 192.168.1.2 (192.168.1.2) 56(84) bytes of data.
64 bytes from 192.168.1.2: icmp_seq=1 ttl=63 time=0.642 ms
64 bytes from 192.168.1.2: icmp_seq=2 ttl=63 time=0.457 ms
64 bytes from 192.168.1.2: icmp_seq=3 ttl=63 time=0.596 ms
...output omitted...
```

- 5.2. While the **ping** command is active, go back to the terminal on **workstation**. As the **developer1** user, pause the running instance.

```
[student@workstation ~] $ openstack server pause \
```

finance-server3

From the instance's console in Firefox, observe the behavior while the instance is paused. Notice that the **ping** command paused, and that the console is not responding.

- 5.3. Back in the terminal on **workstation**, unpause the instance and observe the behavior of the instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server unpause \
finance-server3
```

The **ping** command should resume in the console in Firefox.

- 5.4. Stop the instance and observe the instance's console in Firefox: you should be disconnected.

```
[student@workstation ~ (developer1-finance)]$ openstack server stop \
finance-server3
```

- 5.5. Close Firefox.

- 6. Restart the instance and verify that it is running.

- 6.1. Start the instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server start \
finance-server3
```

- 6.2. List all the available instances, and check the status of the instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server3",
    "Image": "rhel7",
    "ID": "deca6e99-f662-4109-811e-5141c0406564",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.N"
  }
]
```

- 7. Delete the instance and verify it was deleted.

- 7.1. Delete the **finance-server3** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server delete \
finance-server3
```

- 7.2. List all the available instances to verify that the **finance-server3** instance has been deleted.

```
[student@workstation ~ (developer1-finance)]$ openstack server list
```

Cleanup

On **workstation**, run the **lab depinternal-verify cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab depinternal-verify cleanup
```

This concludes the guided exercise.

DEPLOYING AN INSTANCE

PERFORMANCE CHECKLIST

In this lab, you will delete the instance created by the setup script, launch an instance, and verify that the launched instance uses the correct image, flavor and network.

OUTCOMES

You should be able to:

- Delete an instance.
- Create a new instance.
- Verify the newly created instance.

Confirm the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab depinternal-lab** script with the **setup** argument. It creates the resources needed for this lab.

```
[student@workstation ~]$ lab depinternal-lab setup
```

1. On **workstation**, open a terminal as **student** and source the **operator1** environment file, **/home/student/operator1-production-rc**.
2. Delete the existing instance named **production-server6**.
3. Launch an instance named **production-server7** using the **default** flavor, the **rhel7-web** image, and the **production-network1** network.
4. Verify that the instance was created using the correct flavor, network, and image.
5. On **controller0**, verify that you can reach the web server running inside the instance from the DHCP network namespace associated with the **production-network1** network. You can run the **curl** command inside the namespace for that test.
To log in to **controller0** as **root**, you should first log in as the **heat-admin** user and then use the **sudo -i** command to become **root**.

Evaluation

On **workstation**, run the **lab depinternal-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab depinternal-lab grade
```

Cleanup

On **workstation**, run the **lab depinternal-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab depinternal-lab cleanup
```

This concludes the lab.

DEPLOYING AN INSTANCE

PERFORMANCE CHECKLIST

In this lab, you will delete the instance created by the setup script, launch an instance, and verify that the launched instance uses the correct image, flavor and network.

OUTCOMES

You should be able to:

- Delete an instance.
- Create a new instance.
- Verify the newly created instance.

Confirm the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab depinternal-lab** script with the **setup** argument. It creates the resources needed for this lab.

```
[student@workstation ~]$ lab depinternal-lab setup
```

1. On **workstation**, open a terminal as **student** and source the **operator1** environment file, **/home/student/operator1-production-rc**.
 - 1.1. On **workstation**, source the **operator1** user's environment file. The environment file sets the identity service authentication endpoint, user name, password, domain, and project to be used by the OpenStack unified CLI.

```
[student@workstation ~]$ source ~/operator1-production-rc  
[student@workstation ~(operator1-production)]$
```

2. Delete the existing instance named **production-server6**.
 - 2.1. List all available instances to find the instance named **production-server6**.

```
[student@workstation ~(operator1-production)]$ openstack server list -f json  
[  
 {  
 "Status": "ACTIVE",  
 "Name": "production-server6",  
 "Image": "rhel7-web",  
 "ID": "2d7526ed-70bd-400e-94e0-bba213c44f8a",  
 "Flavor": "default",  
 "Networks": "production-network1=192.168.1.N"  
 }  
 ]
```

2.2. Delete the **production-server6** instance.

```
[student@workstation ~ (operator1-production)]$ openstack server delete \
production-server6
```

2.3. List all available instances to verify the instance was deleted.

```
[student@workstation ~ (operator1-production)]$ openstack server list
```

3. Launch an instance named **production-server7** using the **default** flavor, the **rhel7-web** image, and the **production-network1** network.

3.1. Create the **production-server7** instance using the given resources.

```
[student@workstation ~ (operator1-production)]$ openstack server create \
--image rhel7-web \
--flavor default \
--nic net-id=production-network1 \
--wait production-server7
...output omitted...
```

4. Verify that the instance was created using the correct flavor, network, and image.

4.1. View the details of the **production-server7** instance to verify that the correct resources were used.

```
[student@workstation ~ (operator1-production)]$ openstack server show \
production-server7
```

Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2018-06-12T11:21:33.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	production-network1 =192.168.1.P
config_drive	
created	2018-06-12T11:21:01Z
flavor	default (db97...1fbc)
hostId	2e30...efb8
id	16664e74-608d-4d53-9c15-6c317a5cee7a
image	rhel7-web (9c06...cbc7)
key_name	None
name	production-server7
progress	0
project_id	1bfc1e465c0442aab6d5fa387ae1c244
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-12T11:21:34Z
user_id	c0a85a6918b14200884a276d92415872

```
| volumes_attached | | |
```

5. On **controller0**, verify that you can reach the web server running inside the instance from the DHCP network namespace associated with the **production-network1** network. You can run the **curl** command inside the namespace for that test.

To log in to **controller0** as **root**, you should first log in as the **heat-admin** user and then use the **sudo -i** command to become **root**.

- 5.1. Retrieve the IP address of the **production-server7** instance.

```
[student@workstation ~ (operator1-production)]$ openstack server show \  
production-server7
```

Field	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2018-06-12T11:21:33.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	production-network1=192.168.1.P
config_drive	
created	2018-06-12T11:21:01Z
flavor	default (db97...1fbc)
hostId	2e30...efb8
id	1666...ee7a
image	rhel7-web (9c06...cbc7)
key_name	None
name	production-server7
progress	0
project_id	1bfc1e465c0442aab6d5fa387ae1c244
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-12T11:21:34Z
user_id	c0a85a6918b14200884a276d92415872
volumes_attached	

Take a note of the address, **192.168.1.P** in the previous output.

- 5.2. Retrieve the ID of the **production-network1** network. You use this ID in a following step to locate the associated network namespace.

```
[student@workstation ~ (operator1-production)]$ openstack network show \  
production-network1
```

Field	Value
admin_state_up	UP
availability_zone_hints	

availability_zones	nova
created_at	2018-06-12T09:01:11Z
description	
dns_domain	None
id	49f4ab91-42f9-4944-8694-81ba503991b3
ipv4_address_scope	None
ipv6_address_scope	None
is_default	None
is_vlan_transparent	None
mtu	1450
name	production-network1
port_security_enabled	True
project_id	1bfc1e465c0442aab6d5fa387ae1c244
provider:network_type	None
provider:physical_network	None
provider:segmentation_id	None
qos_policy_id	None
revision_number	4
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	dbdb156f-8c2b-4ffa-82ff-d6b4976aed2a
tags	
updated_at	2018-06-12T09:01:29Z

Take a note of the **id**. The value in the previous output, **49f4ab91-42f9-4944-8694-81ba503991b3**, is probably not the same in your environment.

- 5.3. Log in to **controller0** as **heat-admin** and use the **sudo -i** command to become **root**.

```
[student@workstation ~-(operator1-production)]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

- 5.4. List the available DHCP network namespaces and find the one that uses the **production-network1** network ID.

```
[root@controller0 ~]# ip netns
qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 (id: 1)
qdhcp-0177f4ee-87d1-47e0-b0e1-c42171311dce (id: 0)
```

- 5.5. Using the DHCP network namespace for the **production-network1** network, reach the IP address of the **production-server7** instance using the **ping** command.

```
[root@controller0 ~]# ip netns exec \
qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 \
ping -c3 192.168.1.P
PING 192.168.1.P (192.168.1.P) 56(84) bytes of data.
64 bytes from 192.168.1.P: icmp_seq=1 ttl=64 time=5.18 ms
64 bytes from 192.168.1.P: icmp_seq=2 ttl=64 time=3.43 ms
64 bytes from 192.168.1.P: icmp_seq=3 ttl=64 time=4.72 ms
```

```
--- 192.168.1.P ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 3.436/4.449/5.188/0.743 ms
```

5.6. Use the **curl** command to reach the web server running inside the instance.

```
[root@controller0 ~]# ip netns exec \
qdhcp-49f4ab91-42f9-4944-8694-81ba503991b3 \
curl 192.168.1.P
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/
DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
<title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
...output omitted...
```

5.7. Logout from **controller0**.

```
[root@controller0 ~]# exit
[heat-admin@controller0 ~]$ exit
[student@workstation ~](operator1-production)]$
```

Evaluation

On **workstation**, run the **lab depinternal-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab depinternal-lab grade
```

Cleanup

On **workstation**, run the **lab depinternal-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab depinternal-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Administrators can deploy and manage instances using the dashboard or the unified CLI.
- The **openstack network list** command lists the networks available to the current project.
- Instances can be deleted using the **openstack server delete *INSTANCE*** command.
- Running instances can be paused and unpause with the **openstack server pause** and **openstack server unpause** commands.
- Administrators can verify the functionality of an instance and retrieve its parameters with the **show** subcommand of the unified CLI: **openstack object show**.
- Administrators can use the **ip netns** command to troubleshoot network issues and verify the instances connectivity. The network namespace name for a specific network can be retrieved with **openstack network list** command.

CHAPTER 7

MANAGING BLOCK STORAGE

GOAL

Manage ephemeral and persistent block storage.

OBJECTIVES

- Describe cloud storage architecture and features.
- Manage ephemeral block storage.
- Manage persistent block storage.
- Manage snapshots.
- Manage persistent root disks.

SECTIONS

- Describing Features of the Cloud Storage Architecture (and Quiz)
- Managing Ephemeral Block Storage (and Guided Exercise)
- Administering Persistent Block Storage (and Guided Exercise)
- Developing Snapshots (and Guided Exercise)
- Managing Persistent Root Disks (and Guided Exercise)

LAB

- Lab: Managing Block Storage

DESCRIBING FEATURES OF THE CLOUD STORAGE ARCHITECTURE

OBJECTIVES

After completing this section, students should be able to:

- Describe cloud storage architecture and features.

STORAGE IN RED HAT OPENSTACK PLATFORM

A cloud environment application should take advantage of cloud environment benefits, such as leveraging the scalability of compute and storage resources in Red Hat OpenStack Platform. By default, Red Hat OpenStack Platform uses Ceph as the back end for the Block Storage service, but also supports other enterprise-level storage back end products. This support SAN infrastructures, as well as DAS and NAS devices. Customers can reuse of existing enterprise storage infrastructure in OpenStack. Red Hat OpenStack Platform director currently defaults to configuring Red Hat Ceph Storage for as the overcloud's block storage back end.

In a physical enterprise environment, servers are typically installed with direct-attached storage drives, and use external storage for scaling and resource sharing. In cloud-based instances, virtual disks can be directly attached, and external shared storage is provided as a way to scale the local storage. In a self-service cloud environment, storage is a key resource to be managed so that the maximum number of users can take advantage of storage resources.

There are two types of storage: *ephemeral* and *persistent*. Ephemeral storage includes block disk devices and swap space used in a deployed instance. By definition, ephemeral storage resources are discarded when their instance is terminated.

To scale an instance's storage, provision additional virtual disks using the block storage service, object store service, or the file share service. Storage resources provided by these services are *persistent*; they remain after the instance is terminated.

Persistent storage is based on Red Hat OpenStack Platform services which provision this storage to the instances. Red Hat OpenStack Platform supports different storage resources providing persistent storage, including volumes, object containers, and shares.

Volumes

Volumes are the common way to provide persistent storage to instances. They are managed by the block storage service. Like physical machines, volumes are presented as raw devices to the instance's operating system, and can be formatted and mounted for use. A volume in OpenStack can be implemented as different volume types, specified by the backing storage infrastructure or device. A volume can be attached to more than one instance at a time, and can also be moved between instances.

Legacy servers may require that the system (root) disk be persistent. Default ephemeral disk are not able to provide this requirement, but root disks can be created from an existing pre-built, bootable volume. This is possible because Red Hat OpenStack Platform supports the creation of bootable volumes based on images managed by the Image service.

Object Containers

Red Hat OpenStack Platform also includes an object store service which allows to store files as objects. These objects are collected in containers, on which certain access permission can be

configured. This persistent storage is accessible using an API, so it is well-suited for cloud users to upload their data to instances.

Shares

In previous versions of Openstack, a distributed file system had to be created on top of several volumes to share data among several instances at the same time. The file share service (manila) supports the provisioning of shares that can be mounted on several instances at the same time.

BLOCK STORAGE

Block storage uses volumes as its storage unit, which requires the volume to be attached to an instance in order to be accessed. Object storage uses object containers, composed of files and folders, as its storage unit. All objects can be accessed using an API. Object storage does not require an instance to be accessible, but objects can be accessed from inside instances.

Use Cases for Block Storage

The use of block storage in OpenStack depends on the back end storage infrastructure. Depending on back end storage performance, the block storage service can be suitable for high throughput use cases. Currently, the Openstack block storage service supports both Red Hat Ceph Storage and NFS as back ends, and provides drivers allowing native interaction with many common SAN vendors. Volumes are directly served from those infrastructures.

Generically, block storage is well suited for the following use cases:

- Extra space to store large data that needs to be persistent.
- A distributed file system based on raw devices distributed across different instances.
- Back end storage for critical cloud-based applications like distributed databases.

Recommended Practices for Block Storage

In general, the following practices are recommended for block storage in OpenStack:

- Avoid using the LVM as the primary storage in production environments. Red Hat does not support LVM as a primary block storage back end.
- LVM can be used for instance virtual disk management directly on compute nodes.
- Configure a sufficient storage back-end based on workload requirements.
- Configure multiple back ends to use your legacy storage as storage tiers.
- Configure the storage scheduler to allocate volumes on back ends based on volume requirements.



REFERENCES

Further information is available in the Block Storage and Volumes section of the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/storage_guide/

DESCRIBING FEATURES OF THE CLOUD STORAGE ARCHITECTURE

Choose the correct answers to the following questions:

- 1. Which two block storage back ends are supported in Red Hat OpenStack Platform? (Choose two.)
- a. In-memory
 - b. NFS
 - c. Red Hat Ceph Storage
 - d. Raw devices
 - e. LVM
- 2. What is the key difference between ephemeral and persistent storage in Red Hat OpenStack Platform?
- a. Persistent storage supports a broad range of storage infrastructures and ephemeral storage does not.
 - b. Persistent storage always provides worse performance than ephemeral storage.
 - c. Ephemeral storage is deleted when the instance is terminated and persistent storage is not.
 - d. Persistent storage provides fault tolerant storage and ephemeral storage does not.
- 3. Which three of the following storage resources are persistent? (Choose two.)
- a. VM Memory
 - b. Object containers
 - c. Swap memory
 - d. Shares
- 4. Which three use cases are suitable for block storage in Red Hat OpenStack Platform? (Choose three.)
- a. High performance computing
 - b. Back end storage for distributed databases
 - c. Extra space for instances
 - d. Batch processing
 - e. Distributed file systems

▶ **5. What is a recommended practice for block storage?**

- a. Use the LVM back end for production environments.
- b. Deploy most of your instances using volumes as root disks.
- c. Configure multiple back ends for the Block Storage Service, enabling its scheduler to place the volumes on the appropriate back end based on its requirements.
- d. Configure a unique back end to improve performance.

DESCRIBING FEATURES OF THE CLOUD STORAGE ARCHITECTURE

Choose the correct answers to the following questions:

- 1. Which two block storage back ends are supported in Red Hat OpenStack Platform? (Choose two.)
- a. In-memory
 - b. NFS
 - c. Red Hat Ceph Storage
 - d. Raw devices
 - e. LVM
- 2. What is the key difference between ephemeral and persistent storage in Red Hat OpenStack Platform?
- a. Persistent storage supports a broad range of storage infrastructures and ephemeral storage does not.
 - b. Persistent storage always provides worse performance than ephemeral storage.
 - c. Ephemeral storage is deleted when the instance is terminated and persistent storage is not.
 - d. Persistent storage provides fault tolerant storage and ephemeral storage does not.
- 3. Which three of the following storage resources are persistent? (Choose two.)
- a. VM Memory
 - b. Object containers
 - c. Swap memory
 - d. Shares
- 4. Which three use cases are suitable for block storage in Red Hat OpenStack Platform? (Choose three.)
- a. High performance computing
 - b. Back end storage for distributed databases
 - c. Extra space for instances
 - d. Batch processing
 - e. Distributed file systems

► **5. What is a recommended practice for block storage?**

- a. Use the LVM back end for production environments.
- b. Deploy most of your instances using volumes as root disks.
- c. Configure multiple back ends for the Block Storage Service, enabling its scheduler to place the volumes on the appropriate back end based on its requirements.
- d. Configure a unique back end to improve performance.

MANAGING Ephemeral Block Storage

OBJECTIVES

After completing this section, students should be able to:

- Manage ephemeral block storage.

BACK END FILES FOR Ephemeral STORAGE

Ephemeral storage resources for an instance are defined by the flavor used to create the instance. OpenStack flavors currently support the definition of three resources, providing non-persistent storage inside of an instance. Those resources are a root disk, an ephemeral disk, and a swap disk. Each of these resources are mapped as devices to the instances, which the **cloud-init** process configures during the boot process, according to flavor specifications. To properly configure those resources, instances must have access to the metadata service provided by the compute service.

The back-end Ceph RBD images for the different ephemeral storage resources are created on Ceph when instances are deployed. These RBD images use the instance ID as a prefix for their name. The following RBD images are created when an instance is deployed and its associated flavor has a root disk, an ephemeral disk, and swap memory defined.

Back End Files for Ephemeral Storage Resources

FILE NAME	RESOURCE	DESCRIPTION
9d5164a5-e409-4409-b3a0-779e0b90dec9_disk	Root disk	Operating System
9d5164a5-e409-4409-b3a0-779e0b90dec9_disk.eph0	Ephemeral disk	Additional space
9d5164a5-e409-4409-b3a0-779e0b90dec9_disk.swap	Swap disk	Swap memory

When an instance is terminated, the back-end RBD images for associated ephemeral storage resources are deleted. This behavior, which is common in cloud computing environments, contrasts markedly with physical servers, where associated local storage is persistent. This supports the cloud computing concept of self-service access to hardware resources, so unused hardware resources are freed up when they are no longer needed. Instances are designed for use as on-demand processing, with ephemeral storage as a dynamic workspace to facilitate immediate processing. When the processing has finished, the instances and their workspace are no longer needed, by definition, and the ephemeral storage resources are removed.

Ephemeral storage resources for an instance are defined in the flavor used to create that instance. The size for the root disk, ephemeral disk, and swap disk are defined by a flavor. Although defining a root disk size is mandatory, the ephemeral disk and swap disk are optional. If either disk is defined with a size of zero, that disk is created during the instance deploy. Using unnecessarily large sizes for ephemeral and swap disks affects the availability of resources on the compute node where an instance is deployed and the optimal usage of cloud resources.

Root Disk

When the instance is deployed, the root disk is typically created as a copy-on-write clone of the RBD image containing the image used by the instance. The original image is managed by the Image Service, while a copy is stored as a *libvirt base* image on the compute node where the instance is deployed. The root disk is mounted as the first available device in the instance, typically `/dev/vda` or `/dev/sda` in Red Hat Enterprise Linux based instances.

Ephemeral Disk

The ephemeral disk is mapped to the instance as a raw device. Commonly, it is mapped as the second available device, as either `/dev/vdb` or `/dev/sdb` in Red Hat Enterprise Linux based instances. The **cloud-init** process configures this device with a file system, and mounts it in the `/mnt` directory in the instance. The choice of filesystem type and mount point used by **cloud-init** is configurable.

Swap Disk

The swap disk is also mapped to the instance as a raw device. It is mapped to the instance as the next device available, either as `/dev/vdc` or `/dev/sdc` in Red Hat Enterprise Linux based instances when both a root disk and an ephemeral disk are also configured. The **cloud-init** process configures this device as swap and enables it as swap memory in the instance.



NOTE

If the metadata service is not available from the instance, **cloud-init** is unable to prepare the ephemeral and swap disks, but the disks can still be formatted, mounted, and configured manually, as needed.

Managing Ephemeral Storage Using the OpenStack CLI

The following steps outline the process for managing ephemeral storage using the OpenStack unified CLI.

1. Source the environment file to load the credentials for a user.
2. Launch an instance using the **openstack server create** command. Use a flavor that has ephemeral and swap disks configured. Take note of the ID of the instance.
3. Go to the controller node, and display the Ceph pools configured by Red Hat OpenStack Platform director using the **ceph df** command.
4. List the RBD images for the **vms** pool using the **rbd ls --pool vms** command. Three RBD images associated to the instance should be listed. Those RBD images use the ID for the instance as their name prefix, and **_disk** (root disk), **_disk.eph0** (ephemeral disk), and **_disk.swap** (swap disk) as their name suffix.
5. Back in the client machine, obtain the console URL for the instance using the **openstack console url show** command, and open that URL with a web browser.
6. Log in to the instance using the console, and list the ephemeral and swap disks with the **parted -l** command.
7. Verify that the ephemeral disk associated device is mounted in the `/mnt` directory in the instance.
8. Verify that the swap disk associated device is enable as swap using the **swapon -s** command in the instance.
9. Back in the client machine terminal, delete the instance.

10. Log back into the controller node, and confirm that the three RBD images associated to the instance in the **vms** pool have been deleted.



REFERENCES

Further information is available in the Block Storage and Volumes section of the *Storage Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/storage_guide/

MANAGING Ephemeral Block Storage

In this exercise, you will create an instance with ephemeral and swap disks. You will verify that the back end Ceph RBD images associated with those disks have been created. You will also delete the instance and confirm that the associated back end RBD images are removed.

OUTCOMES

You should be able to:

- Create an instance with both swap and ephemeral disks.
- Verify the back end files for both the swap and ephemeral disks.

Confirm the **workstation** and overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as the password. From **workstation**, run **lab blockstorage-ephemeral setup** to verify that the resources created in previous sections are available.

```
[student@workstation ~]$ lab blockstorage-ephemeral setup
```

- ▶ 1. Create an instance, named **finance-server7**, using the **developer1** environment file. Configure this instance to use the **rhel7** image, the **default-extra-disk** flavor, and the **finance-network1** network.
- 1.1. Source the **developer1-finance-rc** environment file to load the **developer1** user credentials.

```
[student@workstation ~]$ source ~/developer1-finance-rc
```

- 1.2. Create an instance, named **finance-server7**, using the **rhel7** image, the **default-extra-disk** flavor, and the **finance-network1** network.

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default-extra-disk \
--nic net-id=finance-network1 \
finance-server7
+-----+-----+
| Field | Value |
+-----+-----+
| ...   |       |
| flavor | default-extra-disk (d7aa...ae28) |
| image  | rhel7 (ca303...da81) |
| name   | finance-server7 |
| ...   |       |
```

- +-----+-----+
- 1.3. Verify that the status of the **finance-server7** instance is **ACTIVE**. It may take some time for the instance to become **ACTIVE**.

```
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-server7 | ACTIVE |
+-----+-----+
```

- ▶ 2. Verify that the swap and the ephemeral disks are available in the **finance-server7** instance.

- 2.1. Show the console URL for the **finance-server7** instance.

```
[student@workstation ~](developer1-finance)]$ openstack console url show \
finance-server7
+-----+-----+
| Field | Value
+-----+-----+
| type | novnc
| url | http://172.25.250.50:6080/vnc_auto.html?token=0a90...a9e6
+-----+-----+
```

- 2.2. Open a Firefox browser, and enter the URL from the previous step to access the **finance-server7** instance console.
- 2.3. Log in to the **finance-server7** instance as **root**, using **redhat** for the password. It may take some time for the console to display the login prompt.
- 2.4. Verify that the swap disk is available in **/dev/vdc**.

```
[root@finance-server7 ~]# swapon -s
Filename          Type      Size    Used   Priority
/dev/vdc          partition 1048572     0      -1
```

- 2.5. Verify that the ephemeral disk is available as **/dev/vdb**, and mounted as **/mnt** in the **finance-server7** instance.

```
[root@finance-server7 ~]# mount | grep /mnt
/dev/vdb on /mnt type vfat (rw,...)
```

- 2.6. Create a 10 MB file named **/mnt/testfile.txt**.

```
[root@finance-server7 ~]# dd \
if=/dev/zero \
of=/mnt/testfile.txt \
bs=1024k \
count=10
10+0 records in
10+0 records out
10485760 bytes (10 MB) copied, 0.0486917 s, 215 MB/s
```

- 3. On **controller0**, find the back end Ceph RBD images for the root disk, the ephemeral disk, and the swap disk for the **finance-server7** instance.

3.1. Retrieve the ID for the **finance-server7** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server show \
finance-server7 -c id -f value
9d5164a5-e409-4409-b3a0-779e0b90dec9
```

3.2. Log in to the **controller0** node as **heat-admin**. No password is required.

```
[student@workstation ~ (developer1-finance)]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$
```

3.3. Display the Ceph pools configured by Red Hat OpenStack Platform director using the **ceph df** command. Verify that the **vms** pool is available.

```
[heat-admin@controller0 ~]$ ceph df
...output omitted...
POOLS:
  NAME      ID      USED      %USED      MAX_AVAIL      OBJECTS
  vms        4       1379M     2.57       52340M          364
...output omitted...
```

3.4. List the RBD images for the **vms** pool using the **rbd ls --pool vms** command. Three RBD images associated to the instance should be listed. Those RBD images use the ID for the **finance-server7** instance as their name prefix, and **_disk** (root disk), **_disk.eph0** (ephemeral disk), and **_disk.swap** (swap disk) as their name suffix.

```
[heat-admin@controller0 ~]$ rbd ls --pool vms
9d5164a5-e409-4409-b3a0-779e0b90dec9_disk
9d5164a5-e409-4409-b3a0-779e0b90dec9_disk.eph0
9d5164a5-e409-4409-b3a0-779e0b90dec9_disk.swap
```

3.5. Log out from **controller0**.

```
[heat-admin@controller0 ~]$ logout
Connection to controller0 closed.
[student@workstation ~ (developer1-finance)]$
```

- 4. On **workstation**, delete the **finance-server7** instance.

4.1. Delete the **finance-server7** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server delete \
finance-server7
```

4.2. Verify that the **finance-server7** instance has been deleted.

```
[student@workstation ~ (developer1-finance)]$ openstack server list
... output omitted ...
```

- 5. Verify that the back end RBD images for the disks associated to the **finance-server7** instance have been removed.

5.1. Log in to the **controller0** server as **heat-admin**. No password is required.

```
[student@workstation ~](developer1-finance)]$ ssh heat-admin@controller0  
[heat-admin@controller0 ~]$
```

- 5.2. List the RBD images for the **vms** pool using the **rbd ls --pool vms** command. No RBD Image using the ID for the **finance-server7** instance as their name prefix should be listed.

```
[heat-admin@controller0 ~]$ rbd ls --pool vms
```

- 5.3. Log out from **controller0**.

```
[heat-admin@controller0 ~]$ logout  
Connection to controller0 closed.  
[student@workstation ~](developer1-finance)]$
```

- 6. Close the browser tab where the **finance-server7** console was displayed.

Cleanup

From **workstation**, run the **lab blockstorage-ephemeral cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab blockstorage-ephemeral cleanup
```

This concludes the guided exercise.

ADMINISTERING PERSISTENT BLOCK STORAGE

OBJECTIVES

After completing this section, students should be able to:

- Manage persistent block storage.

PERSISTENT STORAGE IN RED HAT OPENSTACK PLATFORM

Red Hat OpenStack Platform supports two types of persistent storage: block storage and object storage. Block storage is based on volumes which are provided by the Block Storage service. Object storage is based on object containers provided by the Object Storage service. These containers can include folder and file objects, which are managed using the Object Storage service API. File shares are also available as a technology preview.

PERSISTENT VOLUMES

Block storage, sometimes referred to as volume storage, provides persistent block storage to OpenStack instances. Users attach volumes to running instances. Those volumes can then be formatted with whatever file system is required. This provides a way to have persistent storage attached to the instances. As these volumes are persistent, they can be detached from one instance and reattached to another, and the data remains intact. A volume can also be attached to several instances using the **multattach** volume type.

In Red Hat OpenStack Platform, block storage is provided by the Block Storage service, which supports multiple back ends in the form of drivers. Red Hat OpenStack Platform provides volume drivers for several supported block storage types. These types include:

- iSCSI
- NFS
- Red Hat Ceph Storage

In addition to these drivers, there are drivers available for other major storage vendor systems.

Creating a Volume

Creating a volume does not require administrative privileges. The fields that can be edited are:

Create Volume

FIELD	DESCRIPTION
Volume name	Name of the volume.
Description	Short description of the volume.
Type	Represents a specific storage when multiple back ends are used.
Size (GB)	Volume size (in GB).
Availability zone	Host aggregate to be used for the storage.

Volume Source can also be specified to create the new volume from an existing snapshot, image, or volume. In case this is not specified, it creates an empty volume without any file system or partition table.

To create a volume, go to the Project>Volumes tab and click Create Volume. A volume can also be created with the **openstack volume create** command. The following command creates the **demo-volume1** volume with a size of 1 GB:

```
[user@demo ~(user-demo)]$ openstack volume create --size 1 demo-volume1
```

NOTE

Red Hat does not support LVM (Logical Volume Manager) volumes as primary backend storage for the block storage service, although LVM still has other use cases within OpenStack.

Delete a Volume

The volume can be deleted from the Volumes subtab under Project → Volumes. Select the check boxes for the volumes that need to be deleted. Click Delete Volumes and confirm the choice. When an instance is deleted, the volume is automatically detached from the instance and can be attached to another instance. When an instance is deleted, the data in its attached volumes is not deleted. A volume can also be deleted with the **openstack volume delete** command. The following command deletes the **demo-volume1** volume:

```
[user@demo ~(user-demo)]$ openstack volume delete demo-volume1
```

NOTE

When a volume is deleted, it is erased by being filled with zeros.

Attach a Volume to an Instance

A volume can be attached to a running instance. The status of the volume can be viewed in the Volumes tab of the dashboard. The volume status can be either in **Available** or **In-Use** state. To attach a volume to an instance, go to the Project>Volumes tab and select Manage Attachments in the Actions menu for the volume you want to attach. You can select the instance and the device used to map the volume. A volume can also be attached to an instance with the **openstack server add volume** command. The following command attaches the **demo-volume1** volume to the **demo-server1** instance:

```
[user@demo ~(user-demo)]$ openstack server add volume demo-server1 demo-volume1
```

Users can log in to the instance and mount, format, and use the disk after attaching the volume.

Detach a volume from an instance

To detach a volume, go to the Project>Volumes tab and click Volumes category. The volume can be detached by clicking Manage Attachments and Detach Volume, and confirming the changes. A volume can also be detached from an instance with the **openstack server remove volume** command. The following command detaches the **demo-volume1** volume from the **demo-server1** instance:

```
[user@demo ~(user-demo)]$ openstack server remove volume demo-server1 demo-volume1
```

Use the Volume in an Instance

To use the volume, log in to the instance to format using a file system and mount the volume. Various file systems, such as **ext4** and **xfs**, can be used.

VOLUME TRANSFERS

Some use cases require a volume to be shared between users. This is important for use cases where a volume needs to be shared between different users, like when a cloud administrator needs to configure a bootable volume in a certain way. Only the volumes which are accessible to a user can be transferred by this user. To transfer a volume, the volume must be in the **available** state. This means that the volume must not be attached to any instance at the time the transfer is created.

When the transfer is created, either using CLI or the dashboard, the user transferring the volume gets both a transfer ID and an authorization key. If the transfer is created using the CLI, both the transfer ID and the authorization key are displayed in the command output. If the transfer is created using the dashboard, the transfer ID and the authorization key are available in a text file that can be downloaded when the transfer is created. The transfer ID and the authorization key must be provided to the user receiving the volume transfer in order for the user to be able to accept the transfer. Until the transfer is accepted by the user receiving the volume transfer, the volume is only available to the user creating the transfer. Using this approach, once a transfer is created by a user in a certain project, the volume transfer can be accepted by any other user in any other project by using both the transfer ID and authorization key. When the transfer is accepted, the volume is no longer available to the user creating the transfer, and it becomes available to the user accepting the volume transfer.

The **openstack** command provides several commands to manage a volume transfer, including the creation of a volume transfer, using the **openstack volume transfer request create** command, and the acceptance of a transfer, using the **openstack volume transfer request create accept** command. The dashboard supports volume transfer in the Project → Volumes → Volumes section.

Managing Persistent Storage Using the Dashboard

The following steps outline the process for managing persistent storage using the dashboard, including attaching a volume, detaching a volume, creating a volume transfer, and accepting a volume transfer.

1. Open the dashboard in a web browser, and log in as a user. Navigate to Project → Volumes → Volumes and click Create Volume.

2. Enter a volume name in the Volume Name field. Enter a size for the volume in the Size (GB) field. Click Create Volume.

3. Select Manage Attachments from the list under the Actions column for the previously created volume.

Select the available instance in the Attach to Instance field. Click Attach Volume to attach the volume to the instance.

4. Wait until the Status field for the volume is **In-use**.

5. In the dashboard, navigate to Project → Compute → Instances.

Select Console from the list under the Actions column for the instance.

6. Click Click here to only show console.

7. Log in to the instance using this console.

8. Create a partition in the device associated with the previously identified volume, and format that partition with an XFS file system using the **`mkfs.xfs`** command.
9. Create a directory, and mount the partition on it. Write a file to it.
10. Unmount the partition.
11. Log out from the instance.
12. Navigate to Project → Volumes → Volumes.

Select Manage Attachments from the list under the Actions column for the volume. Click Detach Volume for the instance. Click Detach Volume in the window to confirm.
13. Select Create Transfer from the list under the Actions column for the volume. Enter a transfer name in the Transfer Name field. Click Create Volume Transfer to create the transfer.
14. Click Download transfer credentials and download the file containing both the transfer ID and authorization key. Save the file.
15. Log out of the dashboard, and log in to it using a different user.
16. Open an editor and verify the values of the transfer ID and authorization key in the previously downloaded file.
17. Navigate to Project → Volumes → Volumes and click Accept Transfer.
18. Enter the previously obtained transfer ID in the Transfer ID field. Enter the previously obtained authorization key in the Authorization Key field. Click Accept Volume Transfer to accept the transfer.
19. Verify that the volume status is **Available**.
20. Select Delete Volume from the list under the Actions column for the volume. Click Delete Volume to delete the volume.

Managing Persistent Storage Using the OpenStack CLI

The following steps outlines the process for managing persistent storage using the OpenStack unified CLI, including attaching a volume, detaching a volume, creating a volume transfer, and accepting a volume transfer.

1. Source the environment file to load the authentication parameters for a user.
2. Create a volume using the **`openstack volume create`** command. Configure the size for the volume using the **--size** option.
3. Attach the volume to the available instance using the **`openstack server add volume`** command.
4. Retrieve the URL for the instance console using the **`openstack console url show`** command.
5. Open Firefox and navigate to the instance console URL.
6. Log in to the instance.
7. Create a partition in the device associated with the volume previously identified, and format that partition with a file system (such as an XFS file system using the **`mkfs.xfs`** command).
8. Create a directory, and mount the partition to it. Write a file to it.

9. Back in the terminal with the OpenStack credentials, verify the volume status with the **openstack volume show** command.
10. Try to create a transfer for the volume using the **openstack volume transfer request create** command. It should raise an error because the volume needs to be in the **available** status and it is currently in the **in-use** status.
11. In the instance console, unmount the partition.
12. Back in the terminal with the OpenStack credentials, detach the volume from the instance using the **openstack server remove volume** command.
13. Transfer the volume using the **openstack volume transfer request create** command. The command output returns a transfer ID and authorization key.
14. Source the environment file to load the authentication parameters for another user.
15. Accept the volume transfer using the **openstack volume transfer request accept** command. Use the transfer ID and authorization key previously obtained.
16. List the available volumes using the **openstack volume list** command. The transferred volume should be listed.
17. Delete the volume using the **openstack volume delete** command.



REFERENCES

Further information is available in the Block Storage and Volumes section of the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/storage_guide/

ADMINISTERING PERSISTENT BLOCK STORAGE

In this exercise, you will create a volume, write some data into it, and transfer it to another user. The second user will check that the volume data is still available by mounting the volume in an instance.

OUTCOMES

You should be able to:

- Create a volume.
- Attach a volume to an instance and use it.
- Transfer a volume.
- Verify data persistence in a volume.
- Delete a volume.

Confirm the **workstation** and overcloud's virtual machines are started.

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

From **workstation**, run **lab blockstorage-persistent setup** to verify that OpenStack services are running and that the resources created in previous sections are available. The lab script also creates the **research** project, the **research-server1** instance, and the **developer4** user and its environment file.

```
[student@workstation ~]$ lab blockstorage-persistent setup
```

- ▶ 1. On **workstation**, open Firefox and browse to `http://dashboard.overcloud.example.com`. Log in to the dashboard as **developer1** using **redhat** as a password.
- ▶ 2. As the **developer1** user, create a 1 GB volume named **finance-volume1**
 - 2.1. Navigate to Project → Volumes → Volumes and click +Create Volume. Enter **finance-volume1** in the Volume Name field. Select **1** in the Size (GiB) field. Click Create Volume to create the volume.
- ▶ 3. Attach the **finance-volume1** volume to the **finance-server7** instance, and verify that the device status is **In-Use**.
 - 3.1. Select Manage Attachments in the menu for the **finance-volume1** volume.
 - 3.2. Select the **finance-server7** instance in the Attach to Instance field, then click Attach Volume.

- 3.3. Watch as the Status transitions from **Attaching** to **In-Use**. Verify that the Attached To field sets the **finance-volume1** mapping to **/dev/vdd** on the **finance-server7**.

► 4. Connect to the console of the **finance-server7** instance.

- 4.1. Navigate to Project → Compute → Instances.
Click Console in the menu for the **finance-server7** instance to open its console.
- 4.2. Click **Click here to show only console** link.
- 4.3. Log in to the **finance-server7** instance console using **root** as the user name and **redhat** as the password.

► 5. Partition and mount the **finance-volume1** volume on the **/volume1** directory in the **finance-server7** instance.

- 5.1. Verify that the device associated with the **volume1** volume is available at **/dev/vdd**.

```
[root@finance-server7 ~]# parted /dev/vdd print  
...output omitted...
```

- 5.2. Create a new 1 GB partition on the **/dev/vdd** device.

```
[root@finance-server7 ~]# parted /dev/vdd \  
mklabel msdos \  
mkpart primary xfs 1M 1G  
...output omitted...
```

- 5.3. Create an XFS file system on the **/dev/vdd1** partition.

```
[root@finance-server7 ~]# mkfs.xfs /dev/vdd1  
...output omitted...
```

- 5.4. Create the **/volume1** directory.

```
[root@finance-server7 ~]# mkdir /volume1
```

- 5.5. Mount the **/dev/vdd1** partition on the **/volume1** directory.

```
[root@finance-server7 ~]# mount -t xfs /dev/vdd1 /volume1
```

- 5.6. Verify that the **/dev/vdd1** partition has been correctly mounted.

```
[root@finance-server7 ~]# mount | grep /volume1  
/dev/vdd1 on /volume1 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 5.7. Create a 10 MB file named **/volume1/testfile**

```
[root@finance-server7 ~]# dd \  
if=/dev/zero \  
of=/volume1/testfile \  
bs=1024k \  
count=10  
10+0 records in
```

```
10+0 records out  
10485760 bytes (10 MB) copied, 0.0486917 s, 215 MB/s
```

► 6. Unmount the **volume1** volume from the **finance-server7** instance.

- 6.1. Unmount the **/dev/vdd1** device from **/volume1**.

```
[root@finance-server7 ~]# umount /volume1
```

- 6.2. Verify that the **/dev/vdd1** device has been correctly unmounted. No mount point should be returned for the **/volume1** directory.

```
[root@finance-server7 ~]# mount | grep /volume1
```

► 7. In the dashboard, detach the **volume1** volume from the **finance-server7** instance.

- 7.1. Navigate to Project → Volumes → Volumes and click Manage Attachments in the menu for the **finance-volume1** volume.
- 7.2. Click Detach Volume for the **finance-server7**. Click Detach Volume again in the window to confirm.
- 7.3. Watch as the Status transitions from **Detaching** to **Available** for the volume.

► 8. Transfer the **finance-volume1** volume and save both the transfer ID and authorization key.

- 8.1. Click Create Transfer in the menu for the **finance-volume1** volume.
Enter **financevolume1-transfer** in the Transfer Name field. Click Create Volume Transfer and create the transfer.
- 8.2. Click Download transfer credentials and save the file. This file contains the transfer ID and authorization key. When done, click Close.

► 9. In a terminal on **workstation**, source the **developer4** user credentials. Accept the transfer using the transfer credentials previously obtained.

- 9.1. Source the **developer4-research-rc** file to load the **developer4** user credentials.

```
[student@workstation ~]$ source ~/developer4-research-rc
```

- 9.2. Check the transfer ID and authorization key.

```
[student@workstation ~(developer4-research)]$ cat ~/Downloads/e3170c79-c08c-4a32-8535-7c71b5c26685.txt  
Transfer name: financevolume1-transfer  
Transfer ID: e3170c79-c08c-4a32-8535-7c71b5c26685  
Authorization Key: d160ca8abd86ecb1
```

- 9.3. Accept the volume transfer for the **finance-volume1** volume using the transfer ID and authorization key previously obtained.

```
[student@workstation ~(developer4-research)]$ openstack volume transfer \  
request accept \  
--auth-key d160ca8abd86ecb1 \  
e3170c79-c08c-4a32-8535-7c71b5c26685  
+-----+-----+
```

Field	Value
id	e3170c79-c08c-4a32-8535-7c71b5c26685
name	financevolume1-transfer
volume_id	77b196b6-6bc8-404e-9fbb-5d2cc7b24ede

- 9.4. Verify that the **finance-volume1** volume is now available for the **developer4** user.

```
[student@workstation ~ (developer4-research)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-volume1 | available |
+-----+-----+
```

- 10. Attach the **finance-volume1** volume to the **research-server1** instance created by the lab script.

- 10.1. Verify that the status for the **research-server1** instance is **ACTIVE**.

```
[student@workstation ~ (developer4-research)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| research-server1 | ACTIVE |
+-----+-----+
```

- 10.2. Attach the **finance-volume1** volume to the **research-server1** instance.

```
[student@workstation ~ (developer4-research)]$ openstack server add volume \
research-server1 finance-volume1
```

- 10.3. Verify that the **finance-volume1** has been mapped to the **/dev/vdd** device in the **research-server1** instance.

```
[student@workstation ~ (developer4-research)]$ openstack volume list \
-c Name -c Status -c "Attached to"
+-----+-----+-----+
| Name | Status | Attached to |
+-----+-----+-----+
| finance-volume1 | in-use | Attached to research-server1 on /dev/vdd |
+-----+-----+-----+
```

- 11. In the instance, mount the **/dev/vdd** device on **/volume1**, and verify the data remained intact.

- 11.1. Get the console URL for the **research-server1** instance, and navigate to it using a Firefox browser.

```
[student@workstation ~ (developer4-research)]$ openstack console url show \
research-server1
```

Field	Value
type	novnc
url	http://172.25.250.50:6080/vnc_auto.html?token=f975(...)8b40

- 11.2. Log in to the **research-server1** instance console using **root** as the user name and **redhat** as the password.
- 11.3. Create the **/volume1** directory.

```
[root@research-server1 ~]# mkdir /volume1
```

- 11.4. Mount the **/dev/vdd1** partition on the **/volume1** directory.

```
[root@research-server1 ~]# mount -t xfs /dev/vdd1 /volume1
```

- 11.5. Verify that the **/dev/vdd1** partition has been correctly mounted.

```
[root@research-server1 ~]# mount | grep /volume1
/dev/vdd1 on /volume1 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 11.6. Verify that the **/volume1/testfile** is still available.

```
[root@research-server1 ~]# ls /volume1
testfile
```

► **12.** Unmount the **finance-volume1** volume from the **research-server1** instance.

- 12.1. Unmount the **/dev/vdd1** device from **/volume1**.

```
[root@research-server1 ~]# umount /volume1
```

- 12.2. Verify that the **/dev/vdd1** device has been correctly unmounted. No mount point should be returned for the **/volume1** directory.

```
[root@research-server1 ~]# mount | grep /volume1
```

► **13.** From **workstation**, detach the **finance-volume1** volume from the **research-server1** instance.

- 13.1. Detach the **finance-volume1** volume from the **research-server1** instance.

```
[student@workstation ~(developer4-research)]$ openstack server remove volume \
research-server1 finance-volume1
```

- 13.2. Verify that the **finance-volume1** volume has been correctly detached. Check that the status for the **finance-volume1** volume is **available**.

```
[student@workstation ~(developer4-research)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name      | Status   |
+-----+-----+
```

```
| finance-volume1 | available |  
+-----+-----+
```

► 14. Delete the **finance-volume1** volume.

14.1. Delete the **finance-volume1** volume.

```
[student@workstation ~](developer4-research)]$ openstack volume delete \  
finance-volume1
```

14.2. Verify that the **finance-volume1** volume has been correctly deleted.

```
[student@workstation ~](developer4-research)]$ openstack volume list
```

► 15. Log out of the dashboard and close the browser tab where the instance's console was displayed.

Cleanup

From **workstation**, run the **lab blockstorage-persistent cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab blockstorage-persistent cleanup
```

This concludes the guided exercise.

DEVELOPING SNAPSHOT

OBJECTIVES

After completing this section, students should be able to:

- Manage snapshots.

PERSISTENT BLOCK STORAGE SNAPSHOT

Volume snapshots are taken to preserve the state of a volume at a specific point in time. The volume can be restored from a snapshot in case the volume is deleted.

Similar to snapshots, volume backups can also be taken for a volume. The difference between them is their intended role. Volume backups preserve the data contained in the volume and are used to prevent data loss, whereas snapshots are used to facilitate cloning.

Snapshots can also be used to create a new image from running instances. This is an easy way to upgrade the base image and add customization to an image, but it poses a challenge when dealing with an instance which does not have static data. Taking an instance snapshot of a running instance can lead to inconsistent data. The instance may not boot or run the service as desired. In such cases, the instance file system should be synced and frozen before taking a snapshot, so that persistent snapshots are obtained.

When a snapshot is created, it can be used as a separate volume, which can also be attached to an instance. A volume can have several snapshots associated with it. Note that the space occupied by snapshots counts against the block storage space quota.

Create a Volume Snapshot

To create a snapshot:

1. In the dashboard, navigate to Project → Volumes → Volumes.
2. Select the volume whose snapshot is desired and then select Create Snapshot.
3. Click Create Volume Snapshot after providing a Snapshot Name.

A snapshot can also be created with the **openstack volume snapshot create** command. The following command creates the **demo-snapshot** snapshot for the demo-volume volume.

```
[user@demo ~(user-demo)]$ openstack volume snapshot create \
--volume demo-volume \
demo-snapshot
```

Delete a Volume Snapshot

To delete a snapshot:

1. Click Delete Volume Snapshot from the Actions menu of the snapshot to be deleted.

A snapshot can also be deleted with the **openstack volume snapshot delete** command. The following command deletes the **demo-snapshot** snapshot.

```
[user@demo ~(user-demo)]$ openstack volume snapshot delete \
```

Attach a Snapshot to an Instance

New volumes can be cloned from a snapshot once it appears in the Volume Snapshots table. To do so, select the snapshot from which the volume needs to be cloned, then select Create Volume from the Actions menu. A new volume can also be cloned from a snapshot with the **openstack volume create** command. You can use the **--snapshot** option to specify the source snapshot. The following command creates the **demo-volume** using **demo-snapshot** as the source snapshot.

```
[user@demo ~]# openstack volume create \
--size 1 \
--snapshot demo-snapshot \
demo-volume
```



NOTE

Snapshots no longer need to be detach before creation. The compute service will quiesce the virtual disk before beginning the snapshot.

Managing Snapshots Using the Dashboard

The following steps outline the process for managing snapshots using the dashboard.

1. Open the dashboard in a web browser, and log in as a user.
2. Navigate to Project → Volumes → Volumes and select Manage Attachments in the menu displayed for the available volume.
3. Select the available instance in the Attach to Instance field. Click Attach Volume and attach the volume to the instance. Verify that the device to which the volume has been attached to in the Attached to field.
4. Navigate to Project → Compute → Instances and click Console in the menu for the available instance.
5. Click Click here to only show console.
6. Log in to the instance using this console.
7. Create a partition in the volume associated device previously identified. Format that partition with an XFS file system using the **mkfs.xfs** command.
8. Create a directory, and mount the partition on it. Write some files to it. Umount the partition.
9. Navigate to Project → Volumes → Volumes and select Manage Attachments in the menu displayed for the available volume.
10. Click Detach Volume to detach the volume from the instance. Click Detach Volume again to confirm.
11. Navigate to Project → Volumes → Volumes and click Create Snapshot in the menu displayed for the available volume.
12. Enter the snapshot name in the Snapshot Name field. Click Create Volume Snapshot and create the snapshot.
13. Select Create Volume in the menu displayed for the previous snapshot.

14. Enter the volume name in the **Volume name** field. Click **Create Volume** and create the volume.
15. Select **Manage Attachments** in the menu displayed for the previous volume.
16. Select the available instance in the **Attach to Instance** field. Click **Attach Volume** and attach the volume to the instance. Verify that the device to which the volume has been attached to in the **Attached to** field.
17. Navigate to Project → Compute → Instances and click **Console** in the menu for the available instance.
18. Click **Click here to only show console**.
19. Create a directory, and mount the partition on it. Verify that the files previously created are available.
20. Umount the volume. When done, log out from the instance.
21. Navigate to Project → Volumes → Volumes and select **Manage Attachments** in the menu displayed for the attached volume.
22. Click **Detach Volume** to detach the volume from the instance. Click **Detach Volume** again to confirm.
23. In the dashboard, navigate to Project → Volumes → Volumes, click **Delete Volume** for the snapshot-based volume, and confirm.
24. Click **Delete Volume** again and confirm.
25. In the dashboard, navigate to Project → Volumes → Snapshots, click **Delete Volume Snapshot** for the available snapshot, and confirm.
26. Log out from the dashboard.

Managing Snapshots Using the OpenStack CLI

The following steps outline the process for managing snapshots with the OpenStack unified CLI.

1. Source the environment file to load the authentication parameters for a user.
2. Attach the available volume to the available instance using the **openstack server add volume** command.
3. Retrieve the URL for the instance console using the **openstack console url show** command.
4. Open Firefox and navigate to the instance console URL.
5. Log in to the instance.
6. If not available, create a partition in the volume associated device previously identified, and format that partition with an XFS file system using the **mkfs.xfs** command.
7. Create a directory, and mount the partition on it. Write some files to it. Umount the partition.
8. Detach the volume from the instance using the **openstack server remove volume** command.
9. Create a snapshot for the volume using the **openstack volume snapshot create** command. Configure the source volume using the **--volume** option.

10. Create a volume from the snapshot using the **openstack volume create** command. Configure the snapshot to be used as source of volume using the **--snapshot** option. Configure the snapshot to have the size of the source volume using the **--size** option.
11. Attach the volume to the available instance using the **openstack server add volume** command.
12. Retrieve the URL for the instance console using the **openstack console url show** command.
13. Open Firefox and navigate to the instance console URL.
14. Create a directory, and mount the partition on it. Verify that the files previously created are available.
15. Umount the volume. When done, log out from the instance.
16. Detach the snapshot-based volume from the instance using the **openstack server remove volume** command.
17. Delete the snapshot-based volume using the **openstack volume delete** command.
18. Delete the snapshot using the **openstack volume snapshot delete** command.



REFERENCES

Further information is available in the Block Storage and Volumes section of the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/storage_guide/

DEVELOPING SNAPSHOTS

In this exercise, you will create and delete snapshots. You will also create volumes from snapshots, and attach them to instances.

OUTCOMES

You should be able to:

- Create a volume.
- Create a snapshot.
- Create a volume from a snapshot, and use it in an instance.
- Delete a snapshot.

Confirm that the **workstation** and overcloud's virtual machines are started.

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

From **workstation**, run **lab blockstorage-snapshots setup**, which verifies that OpenStack services are running and the resources created in previous sections are available.

```
[student@workstation ~]$ lab blockstorage-snapshots setup
```

- ▶ 1. On **workstation**, open Firefox and browse to `http://dashboard.overcloud.example.com`. Login to the Dashboard as **developer1** using **redhat** as a password.
- ▶ 2. Attach the **finance-volume1** volume to the **finance-server7** instance.
 - 2.1. Navigate to Project → Volumes → Volumes and select Manage Attachments in the menu for the **finance-volume1** volume.
 - 2.2. Select the **finance-server7** in the Attach to Instance field, then click Attach Volume.
 - 2.3. Verify that the value for the Status field for the **finance-volume1** volume is **In-use**.
- ▶ 3. In the **finance-server7** instance, mount the **finance-volume1** volume to the `/volume1` directory, and add a file.
 - 3.1. Navigate to Project → Compute → Instances and click Console in the menu for the **finance-server7** instance to show the console for this instance.
 - 3.2. Click Click here to show only console.
 - 3.3. Log in to the **finance-server7** as the **root** user using **redhat** as a password.
 - 3.4. Check that the **finance-volume1** volume has been mapped to the `/dev/vdd` device.

```
[root@finance-server7 ~]# parted /dev/vdd print  
...output omitted...
```

- 3.5. Create a new partition in the **/dev/vdd** device.

```
[root@finance-server7 ~]# parted /dev/vdd \  
mklabel msdos \  
mkpart primary xfs 1M 1G  
...output omitted...
```

- 3.6. Create an XFS file system on the **/dev/vdd1** partition.

```
[root@finance-server7 ~]# mkfs.xfs /dev/vdd1  
...output omitted...
```

- 3.7. Create the **/volume1** directory.

```
[root@finance-server7 ~]# mkdir /volume1
```

- 3.8. Mount the **/dev/vdd1** partition on the **/volume1** directory.

```
[root@finance-server7 ~]# mount -t xfs /dev/vdd1 /volume1
```

- 3.9. Verify that the **/dev/vdd1** partition has been correctly mounted.

```
[root@finance-server7 ~]# mount | grep /volume1  
/dev/vdd1 on /volume1 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 3.10. Create a file named **/volume1/testfile.txt** to verify that you can write to the volume.

```
[root@finance-server7 ~]# touch /volume1/testfile.txt
```

- 3.11. Unmount the **/volume1/** directory.

```
[root@finance-server7 ~]# umount /volume1
```

► 4. Detach the **finance-volume1** volume from the **finance-server7** instance.

- 4.1. Navigate to Project → Volumes → Volumes and click Manage Attachments in the menu for the **finance-volume1** volume.
- 4.2. Click Detach Volume to detach the volume from the **finance-server7** instance. Click Detach Volume again to confirm.
- 4.3. Verify that the **finance-volume1** volume status is **Available**. It may take some time for the volume to be detached.

- ▶ 5. Create a snapshot named **finance-volume1-snapshot1** from the **finance-volume1** volume.
 - 5.1. Navigate to Project → Volumes → Volumes and click Create Snapshot in the menu for the **finance-volume1** volume.
 - Enter **finance-volume1-snapshot1** in the Snapshot Name field. Click Create Volume Snapshot. Wait until the value for the Status field for the **finance-volume1-snapshot1** snapshot is **Available**.

- ▶ 6. Create a volume, named **finance-snapshot1-volume1**, from the **finance-volume1-snapshot1** snapshot.
 - 6.1. Click Create Volume in the menu for the **finance-volume1-snapshot1** snapshot.
 - 6.2. Enter **finance-snapshot1-volume1** in the Volume Name field. Click Create Volume.

- ▶ 7. Attach the **finance-snapshot1-volume1** volume to the **finance-server7** instance.
 - 7.1. Select Manage Attachments in the menu for the **finance-snapshot1-volume1** volume.
 - 7.2. Select the **finance-server7** in the Attach to Instance field, then click Attach Volume.
 - 7.3. Verify that the value for the Status field for the **finance-snapshot1-volume1** volume is **In-use**.

- ▶ 8. In the **finance-server7** instance, mount the **finance-snapshot1-volume1** volume to the **/snapshot1-vol1** directory, and verify that the **/snapshot1-vol1/testfile.txt** exists.
 - 8.1. Navigate to Project → Compute → Instances and click Console in the menu for the **finance-server7** instance to show the console for this instance.
 - 8.2. Click Click here to show only console.
 - 8.3. Check that the **finance-snapshot1-volume1** volume has been mapped to the **/dev/vdd** device.

```
[root@finance-server7 ~]# parted /dev/vdd print
...output omitted...
```

- 8.4. Create the **/snapshot1-vol1** directory.

```
[root@finance-server7 ~]# mkdir /snapshot1-vol1
```

- 8.5. Mount the **/dev/vdd1** partition on the **/snapshot1-vol1** directory.

```
[root@finance-server7 ~]# mount -t xfs /dev/vdd1 /snapshot1-vol1
```

- 8.6. Verify that the **/dev/vdd1** partition has been correctly mounted.

```
[root@finance-server7 ~]# mount | grep /snapshot1-vol1
/dev/vdd1 on /snapshot1-vol1 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 8.7. Verify that the **/snapshot1-vol1/testfile.txt** file exists.

```
[root@finance-server7 ~]# ls /snapshot1-vol1/testfile.txt
```

```
/snapshot1-vol1/testfile.txt
```

- 8.8. Unmount the **/snapshot1-vol1/** directory.

```
[root@finance-server7 ~]# umount /snapshot1-vol1
```

- 9. In the CLI, as **developer1**, detach the **finance-snapshot1-volume1** volume from the **finance-server7** instance and delete it.
- 9.1. From **workstation**, open a terminal.
 - 9.2. Source the **developer1-finance-rc** environment file to export the **developer1** user credentials.

```
[student@workstation ~]$ source ~/developer1-finance-rc
```

- 9.3. Detach the **finance-snapshot1-volume1** volume from the **finance-server7** instance.

```
[student@workstation ~-(developer1-finance)]$ openstack server remove volume \
finance-server7 finance-snapshot1-volume1
```

- 9.4. Verify that the **finance-snapshot1-volume1** volume status is **available**. It may take some time for the volume to be detached.

```
[student@workstation ~-(developer1-finance)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-snapshot1-volume1 | available |
| ... | ... |
+-----+-----+
```

- 9.5. Delete the **finance-snapshot1-volume1** volume.

```
[student@workstation ~-(developer1-finance)]$ openstack volume delete \
finance-snapshot1-volume1
```

- 9.6. Verify that the **finance-snapshot1-volume1** volume has been deleted.

```
[student@workstation ~-(developer1-finance)]$ openstack volume list
...output omitted...
```

- 10. Delete the **finance-volume1-snapshot1** snapshot using the **developer1** user credentials.
- 10.1. Delete the **finance-volume1-snapshot1** snapshot.

```
[student@workstation ~-(developer1-finance)]$ openstack volume snapshot delete \

```

finance-volume1-snapshot1

- 10.2. Verify that the **finance-volume1-snapshot1** snapshot has been deleted. No snapshots should display. It may take some time for the snapshot to be deleted.

```
[student@workstation ~](developer1-finance)]$ openstack volume snapshot list
```

- 11. Attach the **finance-volume2** volume to the **finance-server7** instance.

- 11.1. Attach the **finance-volume2** volume to the **finance-server7** instance.

```
[student@workstation ~](developer1-finance)]$ openstack server add volume \
finance-server7 finance-volume2
```

- 11.2. Verify that the **finance-volume2** volume has been correctly attached to the **finance-server7** instance. Confirm that the **finance-volume2** volume status is **in-use** and consult the **Attached to** field for this volume to obtain the device to which it has been mapped to in the **finance-server7** instance. It may take some time for the volume to be attached.

```
[student@workstation ~](developer1-finance)]$ openstack volume list \
-c Name -c Status -c "Attached to"
+-----+-----+-----+
| Name      | Status    | Attached to
+-----+-----+-----+
| finance-volume2 | in-use    | Attached to finance-server7 on /dev/vdd |
| ...       | ...       | ... |
+-----+-----+-----+
```

- 12. In the **finance-server7** instance, mount the **finance-volume2** volume to the **/volume2** directory, and add a file.

- 12.1. Get the console URL for the **finance-server7** instance, and navigate to it using the Firefox browser.

```
[student@workstation ~](developer1-finance)]$ openstack console url show \
finance-server7
+-----+
| Field | Value
+-----+
| type  | novnc
| url   | http://172.25.250.50:6080/vnc_auto.html?token=f7cc...956a
+-----+
```

- 12.2. Check that the **finance-volume2** volume has been mapped to the **/dev/vdd** device.

```
[root@finance-server7 ~]# parted /dev/vdd print
...output omitted...
```

- 12.3. Create a new partition in the **/dev/vdd** device.

```
[root@finance-server7 ~]# parted /dev/vdd \
mklabel msdos \
mkpart primary xfs 1M 1G
```

```
...output omitted...
```

- 12.4. Create an XFS file system on the **/dev/vdd1** partition.

```
[root@finance-server7 ~]# mkfs.xfs /dev/vdd1  
...output omitted...
```

- 12.5. Create the **/volume2** directory.

```
[root@finance-server7 ~]# mkdir /volume2
```

- 12.6. Mount the **/dev/vdd1** partition on the **/volume2** directory.

```
[root@finance-server7 ~]# mount -t xfs /dev/vdd1 /volume2
```

- 12.7. Verify that the **/dev/vdd1** partition has been correctly mounted.

```
[root@finance-server7 ~]# mount | grep /volume2  
/dev/vdd1 on /volume2 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 12.8. Create a file named **/volume2/testfile.txt** to verify that you can write to the volume.

```
[root@finance-server7 ~]# touch /volume2/testfile.txt
```

- 12.9. Unmount the **/volume2** directory.

```
[root@finance-server7 ~]# umount /volume2
```

► 13. On **workstation**, detach the **finance-volume2** volume from the **finance-server7** instance and delete it.

- 13.1. Detach the **finance-volume2** volume from the **finance-server7** instance.

```
[student@workstation ~-(developer1-finance)]$ openstack server remove volume \  
finance-server7 finance-volume2
```

- 13.2. Verify that the **finance-volume2** volume status is **available**. It may take some time for the volume to be detached.

```
[student@workstation ~-(developer1-finance)]$ openstack volume list \  
-c Name -c Status  
+-----+-----+  
| Name      | Status    |  
+-----+-----+  
| finance-volume2 | available |  
| ...       | ...      |  
+-----+-----+
```

- 14. Create a snapshot of the **finance-volume2** volume named **finance-volume2-snapshot1**.
- 14.1. Create a snapshot of the **finance-volume2** volume named **finance-volume2-snapshot1**.

```
[student@workstation ~ (developer1-finance)]$ openstack volume snapshot create \
--volume finance-volume2 \
finance-volume2-snapshot1
+-----+-----+
| Field      | Value           |
+-----+-----+
| created_at | 2018-06-14T17:59:49.127171 |
| description | None            |
| id          | fc3410c1-9060-402c-80ea-6865813a49d1 |
| name        | finance-volume2-snapshot1 |
| properties   |                |
| size         | 1                |
| status        | creating         |
| updated_at   | None             |
| volume_id    | 285f48d2-1025-4270-be7d-72daea700d85 |
+-----+-----+
```

- 14.2. Verify that the **finance-volume2-snapshot1** snapshot has been created by checking its status is **available**.

```
[student@workstation ~ (developer1-finance)]$ openstack volume snapshot list \
-c Name -c Status
+-----+-----+
| Name          | Status     |
+-----+-----+
| finance-volume2-snapshot1 | available |
+-----+-----+
```

- 15. Create a 1 GB volume, named **finance-snapshot2-volume1** from the **finance-volume2-snapshot1** snapshot.
- 15.1. Create a 1 GB volume, named **finance-snapshot2-volume1** from the **finance-volume2-snapshot1** snapshot.

```
[student@workstation ~ (developer1-finance)]$ openstack volume create \
--size 1 \
--snapshot finance-volume2-snapshot1 \
finance-snapshot2-volume1
+-----+-----+
| Field      | Value           |
+-----+-----+
| name        | finance-snapshot2-volume1 |
...output omitted...
+-----+-----+
```

- 15.2. Verify that the **finance-snapshot2-volume1** has been correctly created, checking that its status is **available**.

```
[student@workstation ~ (developer1-finance)]$ openstack volume list \
```

```
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-snapshot2-volume1 | available |
...output omitted...
+-----+-----+
```

- ▶ 16. Attach the **finance-snapshot2-volume1** volume to the **finance-server7** instance.

- 16.1. Attach the **finance-snapshot2-volume1** volume to the **finance-server7** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server add volume \
finance-server7 finance-snapshot2-volume1
```

- 16.2. Verify that the **finance-snapshot2-volume1** volume has been correctly attached to the **finance-server7** instance. Confirm that the **finance-snapshot2-volume1** volume status is **in-use** and verify the **Attached to** field for this volume to obtain the device to which it has been mapped to in the **finance-server7** instance. It may take some time for the volume to be attached.

```
[student@workstation ~ (developer1-finance)]$ openstack volume list \
-c Name -c Status -c "Attached to"
+-----+-----+
| Name | Status | Attached to |
+-----+-----+
| finance-snapshot2-volume1 | in-use | Attached to |
| | | finance-server7 on /dev/vdd |
...output omitted...
+-----+-----+
```

- ▶ 17. In the **finance-server7** instance, mount the **finance-snapshot2-volume1** volume to the **/snapshot2-vol1** directory, and verify that the **/snapshot2-vol1/testfile.txt** exists

- 17.1. Get the console URL for the **finance-server7** instance, and navigate to it using the Firefox browser.

```
[student@workstation ~ (developer1-finance)]$ openstack console url show \
finance-server7
+-----+
| Field | Value |
+-----+
| type | novnc |
| url | http://172.25.250.50:6080/vnc_auto.html?token=f7cc...956a |
+-----+
```

- 17.2. Confirm that the **finance-snapshot2-volume1** volume has been mapped to the **/dev/vdd** device.

```
[root@finance-server7 ~]# parted /dev/vdd print
```

...output omitted...

- 17.3. Create the **/snapshot2-vol1** directory.

```
[root@finance-server7 ~]# mkdir /snapshot2-vol1
```

- 17.4. Mount the **/dev/vdd1** partition on the **/snapshot2-vol1** directory.

```
[root@finance-server7 ~]# mount -t xfs /dev/vdd1 /snapshot2-vol1
```

- 17.5. Verify that the **/dev/vdd1** partition has been correctly mounted.

```
[root@finance-server7 ~]# mount | grep /snapshot2-vol1  
...output omitted...  
/dev/vdd1 on /snapshot2-vol1 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 17.6. Verify that the **/snapshot2-vol1/testfile.txt** file exists.

```
[root@finance-server7 ~]# ls /snapshot2-vol1/testfile.txt  
/snapshot2-vol1/testfile.txt
```

- 17.7. Unmount the **/snapshot2-vol1** directory.

```
[root@finance-server7 ~]# umount /snapshot2-vol1
```

- 18. On **workstation**, detach the **finance-snapshot2-volume1** volume from the **finance-server7** instance.

```
[student@workstation ~(developer1-finance)]$ openstack server remove volume \  
finance-server7 finance-snapshot2-volume1
```

- 19. Verify that the **finance-snapshot2-volume1** volume status is **available**. It may take some time for the volume to be detached.

```
[student@workstation ~(developer1-finance)]$ openstack volume list \  
-c Name -c Status  
+-----+-----+  
| Name | Status |  
+-----+-----+  
| finance-snapshot2-volume1 | available |  
...output omitted...  
+-----+-----+
```

- 20. Log out of the dashboard and close the browser tab where the instance's console was displayed.

Cleanup

From **workstation**, run the **lab blockstorage-snapshots cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab blockstorage-snapshots cleanup
```

This concludes the guided exercise.

MANAGING PERSISTENT ROOT DISKS

OBJECTIVES

After completing this section, you should be able to:

- Manage persistent root disks.

PERSISTENT ROOT DISK USING VOLUMES

By default, instances use ephemeral storage for their root disk so when an instance is terminated, the root disk is removed and no data can be recovered from it. This is expected behavior when using instances in a cloud environment, so resources can be reused by other users of the cloud environment.

There may be use cases where adding persistence to the root disk of the instance is required. This can be done by using a bootable volume as the root disk. Such a bootable volume can be created using an image.



IMPORTANT

If a significant percentage of cloud users use volumes for their root disk, you may run out of available block storage for persistent data storage. Quotas can be used to manage block storage space for projects.

In the dashboard, volumes can be created using images as a source by clicking **Create Volume**, which is available in **Project → Compute → Volumes**. This opens up a dialog with all the volume options, including **Volume Source**, where **Image** can be selected. If **Image** is selected, a new window, named **Use image as a source**, is displayed, where the source image for the volume can be selected. When the volume has been created, an instance can be deployed using it by clicking **Launch Instance** in the dashboard menu. Select the volume to be used as the source disk instead of an image.

Similarly, a volume can be created with the CLI using an image as its source by using the **openstack volume create** command and the **--image** option. When the volume has been created, an instance can be deployed using it by using the **openstack server create** command with the **--volume** option.

The following command deploys the **demo-server1** using the **demo-volume1** volume. It preserves the volume when the instance is terminated.

```
[user@demo ~(user-demo)]$ openstack server create \
--flavor default \
--volume demo-volume1 \
--nic net-id=demo-network1 \
demo-server1
```

SHELVING INSTANCES

Powering off an instance does not free up the resources used by it in the compute node where it is deployed, so those resources cannot be reused. If an instance needs to be stopped for a long time, the Compute service supports shelving instances, so that compute node resources are freed.

When an instance is shelved, a snapshot of the root disk is taken if the instance is not based on a volume. Shelving an instance keeps the instance metadata available to the compute service, so that metadata can be reused to unshelve it. If you redeploy an instance instead of shelving it, the new redeployed instance does not reuse the old metadata, so all the previous instance configuration is lost.

Instance shelving is supported in the dashboard using the **Shelve Instance** and **Unshelve Instance** options in the Actions menu for an instance. In the CLI, instance shelving is supported by the **openstack shelve** and **openstack unshelve** commands. The following command shelves the **demo-server1** instance.

```
[user@demo ~]$(user-demo)]$ openstack server shelve demo-server1
```

The following command unshelves the **demo-server1** instance.

```
[user@demo ~]$(user-demo)]$ openstack server unshelve demo-server1
```

When the instance has been shelved, it shows the **SHELVED_OFFLOADED** status.

Managing Persistent Root Disks Using the Dashboard

The following steps outline the process for managing persistent root disks using the dashboard.

1. Open the dashboard in a web browser, and log in as a user.
2. Navigate to Project → Compute → Images and click Create Volume in the menu for the available image. Ignore the **Unable to retrieve the default volume type** error message.
3. Enter a volume name in the Name field. Enter the source image size in the Size (GiB). Click Create Volume.
4. Navigate to Project → Volumes → Volumes.

Wait until the value for the Status field for the volume is **Available**.

5. In the menu for the previously created volume, click **Launch as Instance**. Enter an instance name in the Instance Name field. In the Flavor section, click + and associate a flavor. Click Launch Instance.
6. Navigate to Project → Compute → Instances.
7. Wait until the value for the Status field for the instance is **Active**.
8. Click **Console** in the menu for the instance to open its console.
9. Click **Click here to show only console**.
10. Log in to the instance console. It may take some time for the console displays the login prompt.
11. Create a file.
12. Navigate to Project → Compute → Instances.
- Click **Delete Instance** in the menu for the instance to delete it, and confirm.
13. Navigate to Project → Volumes → Volumes.

In the menu for the previously created volume, click **Launch as Instance**. Enter an instance name in the Instance Name field. In the Flavor section, click + and associate a flavor. Click Launch Instance.

13. Navigate to Project → Compute → Instances.
Wait until the value for the Status field for the instance is **Active**.
14. Click Console in the menu for the instance to open its console.
15. Click **Click here to show only console**.
16. Log in to the instance console. It may take some time for the console to display the login prompt.
17. Verify that the file previously created is still available.
18. Navigate to Project → Compute → Instances.
Click **Shelve Instance** in the menu for the instance.
19. Wait until the value for the Status field for the instance is **Shelved Offloaded**.
20. Click **Unshelve Instance** in the menu for the instance.
21. Wait until the value for the Status field for the instance is **Active**.
22. Log out from the dashboard.

Managing Persistent Root Disks Using the OpenStack CLI

The following steps outline the process for managing persistent root disk using the OpenStack unified CLI.

1. Source the environment file to load the credentials for a user.
2. Verify that there is an available image.
3. Create a volume from the available images using the **openstack volume create** command. Configure the source image with the **--image** option, and the volume size with the **--size** option.
4. Verify that the volume status is **available** with the **openstack volume list** command.
5. Launch an instance using the **openstack server create** command. Configure the source volume with the **--volume** option.
6. Verify that the instance status is **active** with the **openstack server list** command.
7. Retrieve the URL for the instance console using the **openstack console url show** command.
8. Open Firefox and navigate to the instance console URL.
9. Log in to the instance. It may take some time for the console to display the login prompt.
10. Create a file.
11. Open the **workstation** terminal.
12. Delete the instance using the **openstack server delete** command.

13. Launch an instance using the **openstack server create** command. Configure the source volume with the **--volume** option.
14. Verify that the instance status is **active** with the **openstack server list** command.
15. Retrieve the URL for the instance console using the **openstack console url show** command.
16. Open Firefox and navigate to the instance console URL.
17. Click [Click here to only show console](#).
18. Log in to the instance. It may take some time for the console to display the login prompt.
19. Verify that the file is still available.
20. Open the **workstation** terminal.
21. Shelve the instance using the **openstack server shelve** command. Verify that the status for the instance is **SHELVED_OFFLOADED**.
22. Unshelve the instance using the **openstack server unshelve** command. Verify that the status for the instance is **ACTIVE**.



REFERENCES

Further information is available in the Block Storage and Volumes section of the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/storage_guide/

MANAGING PERSISTENT ROOT DISKS

In this exercise, you will create a bootable volume and demonstrate data persistence using that volume on several instances. You will also demonstrate data persistence shelving and unshelving an instance that uses a bootable volume.

OUTCOMES

You should be able to:

- Create a bootable volume from an image.
- Demonstrate data persistence using a bootable volume.
- Shelve and unshelve an instance that uses a bootable volume.

Confirm the **workstation** and overcloud's virtual machines are started.

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

From **workstation**, run **lab blockstorage-rootdisk setup**, which verifies that the resources created in previous sections are available.

```
[student@workstation ~]$ lab blockstorage-rootdisk setup
```

- ▶ 1. On **workstation**, open Firefox and browse to <http://dashboard.overcloud.example.com>.
- ▶ 2. In the dashboard, login as **developer1**, and create a 10 GB volume from the **rhe17** image, naming it **finance-volume1**.
 - 2.1. Log in to the dashboard as **developer1** using **redhat** as a password.
 - 2.2. Navigate to Project → Compute → Images. Select the dropdown menu for the **rhe17** image and select Create Volume. Ignore the **Unable to retrieve the default volume type** error message.
Enter **finance-volume1** in the Name field. Enter **10** in the Size (GiB). Click +Create Volume.
 - 2.3. Navigate to Project → Volumes → Volumes.
Wait until the value for the Status field for the **finance-volume1** volume is **Available**. You may need to refresh the page for the Status to be updated.
- ▶ 3. Launch an instance named **finance-server8** using the **finance-volume1** volume, the **default** flavor, and the **finance-network1** network.
 - 3.1. In the menu for the **finance-volume1** volume, click **Launch as Instance**. Enter **finance-server8** in the Instance Name field. In the Flavor section, click ↑ that is next to the **default** flavor. In the Networks section, verify that the **finance-network1** network is selected. In the Source section, verify that the **finance-volume1** volume is selected. Click Launch Instance.

- 3.2. Navigate to Project → Compute → Instances.
- Wait until the value for the Status field for the **finance-server8** instance is **Active**. You may need to refresh the web page for the Status to be refreshed. It may take some time for the instance to be active.

► 4. Connect to the console of the instance and create a file.

- 4.1. Click Console in the menu for the **finance-server8** instance to open its console.
- 4.2. Click **Click here to show only console**.
- 4.3. Log in to the **finance-server8** instance console using **root** and **redhat** as a password. It may take some time for the console to display the login prompt.
- 4.4. Create a file called **/root/testfile.txt**.

```
[root@finance-server8 ~]# touch testfile.txt
```

► 5. In the CLI, delete the **finance-server8** instance using the **developer1** user credentials.

- 5.1. On **workstation**, source the **developer1-finance-rc** file to load the **developer1** user credentials.

```
[student@workstation ~]$ source ~/developer1-finance-rc
```

- 5.2. Delete the **finance-server8** instance.

```
[student@workstation ~(developer1-finance)]$ openstack server delete \
finance-server8
```

- 5.3. Verify that the **finance-server8** instance has been correctly deleted. It may take some time for the instance to be removed.

```
[student@workstation ~(developer1-finance)]$ openstack server list
...output omitted...
```

► 6. Create a new instance named **finance-server9** using the **finance-volume1** volume and the **default** flavor.

- 6.1. Create a new instance named **finance-server9** using the **finance-volume1** volume, the **default** flavor, and the **finance-network1** network.

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--volume finance-volume1 \
--nic net-id=finance-network1 \
finance-server9
+-----+
| Field           | Value          |
+-----+
...output omitted...
| flavor          | default (12e28c2e-7e28-4e18-bba0-7782496398aa) |
| name            | finance-server9 |
...output omitted...
```

- 6.2. Verify that the status of the **finance-server9** instance is **ACTIVE**. It may take some time for the status to be **ACTIVE**.

```
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-server9 | ACTIVE |
+-----+-----+
```

- 7. Connect to the console of the **finance-server9** instance. Confirm that the previously created **/root/testfile.txt** file is persistent.

- 7.1. Retrieve the console URL for the **finance-server9** instance.

```
[student@workstation ~](developer1-finance)]$ openstack console url show \
finance-server9
+-----+-----+
| Field | Value
+-----+-----+
| type | novnc
| url | http://172.25.250.50:6080/vnc_auto.html?token=de97...5c62
+-----+-----+
```

- 7.2. Open Firefox and navigate to the console URL.
7.3. Log in to the **finance-server9** instance console using **root** and **redhat** as a password. It may take some time for the console to display the login prompt.
7.4. Verify that the **/root/testfile.txt** file is persistent.

```
[root@finance-server9 ~]# ls testfile.txt
testfile.txt
```

- 8. Delete the **finance-server9** instance and the **finance-volume1** volume.

- 8.1. On **workstation**, delete the **finance-server9** instance.

```
[student@workstation ~](developer1-finance)]$ openstack server delete \
finance-server9
```

- 8.2. Verify that the **finance-server9** instance has been correctly deleted.

```
[student@workstation ~](developer1-finance)]$ openstack server list
...output omitted...
```

- 8.3. Delete the **finance-volume1** volume.

```
[student@workstation ~](developer1-finance)]$ openstack volume delete \
```

finance-volume1

- 8.4. Verify that the **finance-volume1** volume has been correctly deleted. The volume **finance-volume1** should not be listed.

```
[student@workstation ~ (developer1-finance)]$ openstack volume list
```

- 9. Create a 10 GB volume from the **rhel7** image named **finance-volume2**.

- 9.1. Create a volume from the **rhel7** image named **finance-volume2**. Configure the size of this volume to be 10 GB.

```
[student@workstation ~ (developer1-finance)]$ openstack volume create \
--size 10 \
--image rhel7 \
finance-volume2
+-----+-----+
| Field | Value |
+-----+-----+
...output omitted...
| name | finance-volume2 |
| size | 10 |
...output omitted...
+-----+-----+
```

- 9.2. Verify that the **finance-volume2** volume has been correctly created. Check that the value for the **Status** field is **available**. It may take some time for the volume to be **available**.

```
[student@workstation ~ (developer1-finance)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-volume2 | available |
+-----+-----+
```

- 10. Launch an instance named **finance-server10** using the **finance-volume2** volume and other options you used previously.

- 10.1. Launch an instance named **finance-server10** using the **finance-volume2** volume.

```
[student@workstation ~ (developer1-finance)]$ openstack server create \
--flavor default \
--nic net-id=finance-network1 \
--volume finance-volume2 \
finance-server10
+-----+-----+
| Field | Value |
+-----+-----+
...output omitted...
| flavor | default (12e28c2e-7e28-4e18-bba0-7782496398aa) |
| name | finance-server10 |
...output omitted...
```

- +-----+
10.2. Verify that the status of the **finance-server10** instance is **ACTIVE**. It may take some time for the status to be **ACTIVE**.

```
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-server10 | ACTIVE |
+-----+-----+
```

- 11. Connect to the console of the **finance-server10** instance and create the **/root/testfile.txt** file.

- 11.1. Retrieve the console URL for the **finance-server10** instance.

```
[student@workstation ~](developer1-finance)]$ openstack console url show \
finance-server10
+-----+-----+
| Field | Value |
+-----+-----+
| type | novnc |
| url | http://172.25.250.50:6080/vnc_auto.html?token=c851...d75e |
+-----+-----+
```

- 11.2. Open Firefox and navigate to the console URL.

- 11.3. Log in to the **finance-server10** instance console using **root** and **redhat** as a password. It may take some time for the console to display the login prompt.

- 11.4. Create the **/root/testfile.txt** file.

```
[root@finance-server10 ~]# touch testfile.txt
```

- 11.5. Verify that the **/root/testfile.txt** file has been correctly created.

```
[root@finance-server10 ~]# ls testfile.txt
testfile.txt
```

- 12. On **workstation**, shelve the **finance-server10** instance, and verify that it is in the **SHELVED_OFFLOADED** status.

- 12.1. Shelve the **finance-server10** instance.

```
[student@workstation ~](developer1-finance)]$ openstack server shelve \
finance-server10
```

- 12.2. Verify that the **finance-server10** instance status is **SHELVED_OFFLOADED**.

```
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
```

```
| finance-server10 | SHELVED_OFFLOADED |  
+-----+-----+
```

► 13. Unshelve the **finance-server10** instance.

- 13.1. Unshelve the **finance-server10** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server unshelve \  
finance-server10
```

- 13.2. Verify that the **finance-server10** instance has been correctly unshelved. Check that the value for the **Status** field is **ACTIVE**.

```
[student@workstation ~ (developer1-finance)]$ openstack server list \  
-c Name -c Status  
+-----+-----+  
| Name | Status |  
+-----+-----+  
| finance-server10 | ACTIVE |  
+-----+-----+
```

► 14. Connect to the console of the **finance-server10** instance and verify that the **/root/testfile.txt** file is still available.

- 14.1. Retrieve the console URL for the **finance-server10** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack console url show \  
finance-server10  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| type | novnc |  
| url | http://172.25.250.50:6080/vnc_auto.html?token=fb21...882c |  
+-----+-----+
```

- 14.2. Open Firefox and navigate to the console URL.
- 14.3. Log in to the **finance-server10** instance console using **root** and **redhat** as a password. It may take some time for the console to display the login prompt.
- 14.4. Verify that the **/root/testfile.txt** file is still available.

```
[root@finance-server10 ~]# ls testfile.txt  
testfile.txt
```

► 15. On **workstation**, delete the **finance-server10** instance.

- 15.1. Delete the **finance-server10** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server delete \  
finance-server10
```

- 15.2. Verify that the **finance-server10** instance has been correctly deleted. It may take some time for the instance to be deleted.

```
[student@workstation ~ (developer1-finance)]$ openstack server list
```

...output omitted...

- 16. Log out of dashboard and close the browser tab where the instance's console was displayed.

Cleanup

From **workstation**, run the **lab blockstorage-rootdisk cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab blockstorage-rootdisk cleanup
```

This concludes the guided exercise.

MANAGING BLOCK STORAGE

PERFORMANCE CHECKLIST

In this lab, you will create volumes, attach them to instances, and snapshot them. You will also create volumes from images and use them to boot instances. Finally you will transfer volumes between users, and shelve instances.

OUTCOMES

You should be able to:

- Create a volume.
- Attach a volume to an instance.
- Create a snapshot.
- Transfer a volume from one user to another.
- Create a volume from an image.
- Boot an instance using a volume.
- Shelve an instance.

Confirm that the **workstation** and overcloud's virtual machines are started.

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

From **workstation**, run **lab blockstorage-lab setup**, which verifies that OpenStack services are running, and creates the **production-volume2** volume, and the **consulting-volume1** instance.

```
[student@workstation ~]$ lab blockstorage-lab setup
```

1. Create a volume named **production-volume1** using the **operator1** user credentials. Configure the size of this volume to be 1 GB.
2. Attach the **production-volume1** volume to the **production-server1** instance created by the lab script.
3. As the **operator1** user, open a console to the **production-server1** instance.
4. Create an XFS file system on the **production-volume1** volume.
5. Mount the previous file system on the **/volume1** directory in the **production-server1** instance, and create a file named **/volume1/testfile.txt**. When done, unmount the file system.
6. Detach the **production-volume1** volume from the **production-server1** instance.
7. Create a snapshot for the **production-volume1** volume. Name it **production-volume1-snapshot1**.

8. Create a volume named **production-snapshot1-volume1** using the **production-volume1-snapshot1** snapshot. Configure the size of this volume to be 1 GB.
9. Attach the **production-snapshot1-volume1** volume to the **production-server1** instance.
10. Open a console to the **production-server1** instance.
11. Mount the previous file system on the **/snapshot** directory at the **production-server1** instance, and verify that the **/snapshot/testfile.txt** file exists. When done, unmount the file system.
12. Detach the **production-snapshot1-volume1** volume from the **production-server1** instance.
13. Request a volume transfer for the **consulting-volume1** volume (created by the lab script) as **operator2** user from the **consulting** project.
14. Accept the volume transfer using the **operator1** user.
15. Create a volume from the **rhel7** image named **production-volume3**. Configure the size of this volume to be 10 GB.
16. Create an instance named **production-server2** using the **production-volume3** volume, the **production-network1** network, and the **default** flavor.
17. Shelf the **production-server2** instance.

Evaluation

On **workstation**, run the **lab blockstorage-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab blockstorage-lab grade
```

Cleanup

From **workstation**, run the **lab blockstorage-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab blockstorage-lab cleanup
```

This concludes the lab.

MANAGING BLOCK STORAGE

PERFORMANCE CHECKLIST

In this lab, you will create volumes, attach them to instances, and snapshot them. You will also create volumes from images and use them to boot instances. Finally you will transfer volumes between users, and shelve instances.

OUTCOMES

You should be able to:

- Create a volume.
- Attach a volume to an instance.
- Create a snapshot.
- Transfer a volume from one user to another.
- Create a volume from an image.
- Boot an instance using a volume.
- Shelve an instance.

Confirm that the **workstation** and overcloud's virtual machines are started.

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

From **workstation**, run **lab blockstorage-lab setup**, which verifies that OpenStack services are running, and creates the **production-volume2** volume, and the **consulting-volume1** instance.

```
[student@workstation ~]$ lab blockstorage-lab setup
```

1. Create a volume named **production-volume1** using the **operator1** user credentials. Configure the size of this volume to be 1 GB.
 - 1.1. Source the **operator1-production-rc** environment file to export the **operator1** user credentials.

```
[student@workstation ~]$ source ~/operator1-production-rc
```

- 1.2. Create a volume named **production-volume1**. Configure the size of this volume to be 1 GB.

```
[student@workstation ~]$ openstack volume create \
--size 1 production-volume1
+-----+-----+
| Field | Value |
+-----+-----+
...output omitted...
```

```
| name           | production-volume1          |
...output omitted...
+-----+-----+
```

2. Attach the **production-volume1** volume to the **production-server1** instance created by the lab script.

- 2.1. Verify that the **production-server1** status is **ACTIVE**.

```
[student@workstation ~(operator1-production)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name      | Status |
+-----+-----+
| production-server1 | ACTIVE |
+-----+-----+
```

- 2.2. Attach the **production-volume1** volume to the **production-server1** instance.

```
[student@workstation ~(operator1-production)]$ openstack server add volume \
production-server1 production-volume1
```

- 2.3. Verify that the **production-volume1** has been correctly attached to the **production-server1** instance as the **/dev/vdb** device.

```
[student@workstation ~(operator1-production)]$ openstack volume list \
-c Name -c "Attached to"
+-----+-----+
| Name      | Attached to          |
+-----+-----+
| production-volume1 | Attached to production-server1 on /dev/vdb |
...output omitted...
+-----+-----+
```

3. As the **operator1** user, open a console to the **production-server1** instance.

- 3.1. Retrieve the console URL for the **production-server1** instance.

```
[student@workstation ~(operator1-production)]$ openstack console url show \
production-server1
+-----+
| Field | Value
+-----+
| type  | novnc
| url   | http://172.25.250.50:6080/vnc_auto.html?token=d502(....)4c27 |
+-----+
```

- 3.2. Open a Firefox browser and navigate to the console URL previously obtained.
3.3. Click [Click here](#) to show only console.
3.4. Log in to the **production-server1** using **root** and **redhat** as a password. It may take some time for the console to display the login prompt.

4. Create an XFS file system on the **production-volume1** volume.

- 4.1. Verify in the **production-server1** instance that the **production-volume1** volume is mapped to the **/dev/vdb** device.

```
[root@production-server1 ~]# parted /dev/vdb print
Error: /dev/vdb: unrecognised disk label
Model: Virtio Block Device (virtblk)
Disk /dev/vdb: 1074MB
Sector size (logical/physical): 512B/512B
Partition Table: unknown
Disk Flags:
```

- 4.2. Create a new partition in the **/dev/vdb** device.

```
[root@production-server1 ~]# parted /dev/vdb \
mklabel msdos \
mkpart primary xfs 1M 1G
...output omitted...
```

- 4.3. Create an XFS file system on the **/dev/vdb1** partition.

```
[root@production-server1 ~]# mkfs.xfs /dev/vdb1
...output omitted...
```

5. Mount the previous file system on the **/volume1** directory in the **production-server1** instance, and create a file named **/volume1/testfile.txt**. When done, unmount the file system.

- 5.1. Create the **/volume1** directory.

```
[root@production-server1 ~]# mkdir /volume1
```

- 5.2. Mount the **/dev/vdb1** partition on the **/volume1** directory.

```
[root@production-server1 ~]# mount -t xfs /dev/vdb1 /volume1
```

- 5.3. Verify that the **/dev/vdb1** partition has been correctly mounted.

```
[root@production-server1 ~]# mount | grep /volume1
/dev/vdb1 on /volume1 type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

- 5.4. Create a file named **/volume1/testfile.txt**

```
[root@production-server1 ~]# touch /volume1/testfile.txt
```

- 5.5. Unmount the **/dev/vdb1** partition from the **/volume1** directory.

```
[root@production-server1 ~]# umount /volume1
```

6. Detach the **production-volume1** volume from the **production-server1** instance.

- 6.1. On **workstation**, detach the **production-volume1** volume from the **production-server1** instance.

```
[student@workstation ~](operator1-production)]$ openstack server remove volume \
```

production-server1 production-volume1

- 6.2. Verify that the **production-volume1** volume has been correctly detached. Check that the status for the **production-volume1** volume is **available**.

```
[student@workstation ~(operator1-production)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| production-volume1 | available |
...output omitted...
+-----+-----+
```

7. Create a snapshot for the **production-volume1** volume. Name it **production-volume1-snapshot1**.

- 7.1. Create a snapshot for the **production-volume1** volume. Name it **production-volume1-snapshot1**.

```
[student@workstation ~(operator1-production)]$ openstack volume \
snapshot create \
--volume production-volume1 \
production-volume1-snapshot1
+-----+-----+
| Field | Value |
+-----+-----+
...output omitted...
| name | production-volume1-snapshot1 |
...output omitted...
+-----+-----+
```

- 7.2. Verify that the status for the **production-volume1-snapshot1** snapshot is **available**.

```
[student@workstation ~(operator1-production)]$ openstack volume snapshot list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| production-volume1-snapshot1 | available |
+-----+-----+
```

8. Create a volume named **production-snapshot1-volume1** using the **production-volume1-snapshot1** snapshot. Configure the size of this volume to be 1 GB.

- 8.1. Create a volume, named **production-snapshot1-volume1**, using the **production-volume1-snapshot1** snapshot. Configure the size of this volume to be 1 GB.

```
[student@workstation ~(operator1-production)]$ openstack volume create \
--size 1 \
--snapshot production-volume1-snapshot1 \
production-snapshot1-volume1
+-----+-----+
| Field | Value |
+-----+-----+
```

```
+-----+  
...output omitted...  
| name | production-snapshot1-volume1 |  
| size | 1 |  
...output omitted...  
+-----+
```

- 8.2. Verify that the status for the **production-snapshot1-volume1** volume is **available**.

```
[student@workstation ~ (operator1-production)]$ openstack volume list \  
-c Name -c Status  
+-----+-----+  
| Name | Status |  
+-----+-----+  
| production-snapshot1-volume1 | available |  
...output omitted...  
+-----+
```

9. Attach the **production-snapshot1-volume1** volume to the **production-server1** instance.

- 9.1. Attach the **production-snapshot1-volume1** volume to the **production-server1** instance.

```
[student@workstation ~ (operator1-production)]$ openstack server add volume \  
production-server1 production-snapshot1-volume1
```

- 9.2. Verify that the **production-snapshot1-volume1** has been correctly attached to the **production-server1** instance as the **/dev/vdb** device.

```
[student@workstation ~ (operator1-production)]$ openstack volume list \  
-c Name -c "Attached to"  
+-----+-----+  
| Name | Attached to |  
+-----+-----+  
| production-snapshot1-volume1 | Attached to production-server1 on /dev/vdb |  
...output omitted...  
+-----+
```

10. Open a console to the **production-server1** instance.

- 10.1. Retrieve the console URL for the **production-server1** instance.

```
[student@workstation ~ (operator1-production)]$ openstack console url show \  
production-server1  
+-----+  
| Field | Value |  
+-----+  
| type | novnc |  
| url | http://172.25.250.50:6080/vnc_auto.html?token=d502(....)4c27 |  
+-----+
```

- 10.2. Open a Firefox browser and navigate to the console URL previously obtained.

11. Mount the previous file system on the **/snapshot** directory at the **production-server1** instance, and verify that the **/snapshot/testfile.txt** file exists. When done, unmount the file system.

11.1. Create the **/snapshot** directory.

```
[root@production-server1 ~]# mkdir /snapshot
```

11.2. Mount the **/dev/vdb1** partition on the **/snapshot** directory.

```
[root@production-server1 ~]# mount -t xfs /dev/vdb1 /snapshot
```

11.3. Verify that the **/dev/vdb1** partition has been correctly mounted.

```
[root@production-server1 ~]# mount | grep /snapshot  
/dev/vdb1 on /snapshot type xfs (rw,relatime,seclabel,attr2,inode64,noquota)
```

11.4. Verify that the **/snapshot/testfile.txt** file exists.

```
[root@production-server1 ~]# ls /snapshot/testfile.txt  
/snapshot/testfile.txt
```

11.5. Unmount the **/dev/vdb1** partition from the **/snapshot** directory.

```
[root@production-server1 ~]# umount /snapshot
```

12. Detach the **production-snapshot1-volume1** volume from the **production-server1** instance.

12.1. On **workstation**, detach the **production-snapshot1-volume1** volume from the **production-server1** instance.

```
[student@workstation ~(operator1-production)]$ openstack server remove volume \  
production-server1 production-snapshot1-volume1
```

12.2. Verify that the **production-snapshot1-volume1** volume has been correctly detached. Check that the status for the **production-snapshot1-volume1** volume is **available**.

```
[student@workstation ~(operator1-production)]$ openstack volume list \  
-c Name -c Status  
+-----+-----+  
| Name | Status |  
+-----+-----+  
| production-snapshot1-volume1 | available |  
...output omitted...  
+-----+-----+
```

13. Request a volume transfer for the **consulting-volume1** volume (created by the lab script) as **operator2** user from the **consulting** project.

13.1. Source the **operator2-consulting-rc** file to load the **operator2** user credentials.

```
[student@workstation ~(operator1-production)]$ source ~/operator2-consulting-rc
```

```
[student@workstation ~ (operator2-consulting)]$
```

- 13.2. Verify that the **consulting-volume1** volume is in the **available** status.

```
[student@workstation ~ (operator2-consulting)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| consulting-volume1 | available |
+-----+-----+
```

- 13.3. Create a volume transfer for the **consulting-volume1** volume. Set the name of the transfer to **consultingvolume1-transfer**. Make a note of the **auth_key** and **id** fields.

```
[student@workstation ~ (operator2-consulting)]$ openstack volume transfer \
request create \
--name consultingvolume1-transfer \
consulting-volume1
+-----+-----+
| Field | Value |
+-----+-----+
| auth_key | b30da8b74e915ca4 |
| id | 4c5b9156-6dbf-4528-a41c-0605ab77aaa9 |
| name | consultingvolume1-transfer |
...output omitted...
+-----+-----+
```

14. Accept the volume transfer using the **operator1** user.

- 14.1. Source the **operator1-production-rc** file to load the **operator1** user credentials.

```
[student@workstation ~ (operator2-consulting)]$ source ~/operator1-production-rc
[student@workstation ~ (operator1-production)]$
```

- 14.2. Accept the transfer by using both the values for the **auth_key** and **id** fields previously obtained.

```
[student@workstation ~ (operator1-production)]$ openstack volume transfer \
request accept \
--auth-key b30da8b74e915ca4 \
4c5b9156-6dbf-4528-a41c-0605ab77aaa9
+-----+-----+
| Field | Value |
+-----+-----+
| id | 4c5b9156-6dbf-4528-a41c-0605ab77aaa9 |
| name | consultingvolume1-transfer |
...output omitted...
+-----+-----+
```

- 14.3. Verify that the **consulting-volume1** volume is available for the **operator1** user.

```
[student@workstation ~ (operator1-production)]$ openstack volume list \
-c Name -c Status
```

```
+-----+-----+
| Name | Status |
+-----+-----+
| consulting-volume1 | available |
...output omitted...
+-----+-----+
```

15. Create a volume from the **rhel7** image named **production-volume3**. Configure the size of this volume to be 10 GB.

- 15.1. Verify that the **rhel7** image is active.

```
[student@workstation ~-(operator1-production)]$ openstack image list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| rhel7 | active |
+-----+-----+
```

- 15.2. Create a volume from the **rhel7** image named **production-volume3**. Configure the size of this volume to be 10 GB.

```
[student@workstation ~-(operator1-production)]$ openstack volume create \
--size 10 \
--image rhel7 \
production-volume3
+-----+-----+
| Field | Value |
+-----+-----+
...output omitted...
| name | production-volume3 |
| size | 10 |
...output omitted...
+-----+-----+
```

16. Create an instance named **production-server2** using the **production-volume3** volume, the **production-network1** network, and the **default** flavor.

- 16.1. Verify that the **production-volume3** volume is available. It may take some time for this volume to become available.

```
[student@workstation ~-(operator1-production)]$ openstack volume list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| production-volume3 | available |
...output omitted...
+-----+-----+
```

- 16.2. Create an instance named **production-server2** using the **production-volume3** volume, the **production-network1** network, and the **default** flavor.

```
[student@workstation ~-(operator1-production)]$ openstack server create \
--nic net-id=production-network1 \
```

```
--flavor default \
--volume production-volume3 \
production-server2
+-----+
| Field | Value |
+-----+
...output omitted...
| flavor | default (12e28c2e-7e28-4e18-bba0-7782496398aa) |
| name | production-server2 |
...output omitted...
+-----+
```

- 16.3. Verify that the status of the **production-server2** instance is **ACTIVE**. It may take some time for the status to be **active**.

```
[student@workstation ~](operator1-production)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| production-server2 | ACTIVE |
...output omitted...
+-----+-----+
```

17. Shelve the **production-server2** instance.

- 17.1. Shelve the **production-server2** instance.

```
[student@workstation ~](operator1-production)]$ openstack server shelve \
production-server2
```

- 17.2. Verify that the **production-server2** instance status is **SHELVED_OFFLOADED**.

```
[student@workstation ~](operator1-production)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| production-server2 | SHELVED_OFFLOADED |
...output omitted...
+-----+-----+
```

Evaluation

On **workstation**, run the **lab blockstorage-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab blockstorage-lab grade
```

Cleanup

From **workstation**, run the **lab blockstorage-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab blockstorage-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Red Hat OpenStack Platform supports both ephemeral and persistent storage. Ephemeral storage is removed when the instance is terminated.
- Ephemeral storage includes root disks, ephemeral disks, and swap disks.
- Persistent storage includes volumes, object containers, and shares.
- Block storage within Red Hat OpenStack Platform is provided by the Block Storage Service using one of the supported storage back ends, such as NFS or Red Hat Ceph Storage.
- The Block Storage Service provides persistent storage with volumes.
- Snapshots can be used to preserve a point-in-time state of a volume.
- The root disk of an instance can be persistent by using a bootable volume as root disk.

CHAPTER 8

MANAGING OBJECT STORAGE

GOAL

Manage Object Storage

OBJECTIVES

- Describe objects and object storage architecture.
- Manage objects.

SECTIONS

- Describing Object Storage Architecture (and Quiz)
- Managing Objects (and Guided Exercise)

LAB

- Managing Object Storage



DESCRIBING OBJECT STORAGE ARCHITECTURE

OBJECTIVES

After completing this section, students should be able to describe objects and object storage architecture.

WHAT IS AN OBJECT CONTAINER?

A Container Stores Objects

An object container is a storage compartment for objects. Containers provide a way to organize objects. Containers are like directories in Red Hat Enterprise Linux except that containers cannot be nested. Users can create an unlimited number of containers within their account. A container can be made public, allowing anyone with the Public URL access to objects in the container.

WHAT IS AN OBJECT?

An Object is Data

In terms of object storage, an object is the stored data. An object is stored as a binary file along with metadata which is stored in the file's extended attributes (xattrs). An object can be many different types of data such as text files, videos, images, emails, or virtual machine images.

Pseudo-folders

Another type of object is a pseudo-folder object which can be used to simulate a hierarchical structure for better organization of objects in the container. For example, a container named *books* has a pseudo-folder within it named *computer*. Within the computer folder there is another pseudo-folder named *linux*. Within *linux* there are two file objects, one named *the-introduction-to-linux-book.pdf* and the other named *the-introduction-to-linux-book.epub*. The container structure would look similar to the following:

```
books
  computer/linux/the-introduction-to-linux-book.pdf
  computer/linux/the-introduction-to-linux-book.epub
```

Object Ownership

When an object is created, it is owned by an account, such as an OpenStack project. The account service uses a database to track containers owned by that account.

Containers are where the objects are stored. The container service uses a database to track objects owned by the container. The object service uses a database to track and store container objects.

Accessing an Object

Creating or retrieving an object is handled by a proxy service which uses a hash of the path to the object to create a unique object ID. Users and applications use the object's ID to access the object.

Expanded details on these concepts and processes are provided later in this chapter.

WHAT IS OBJECT STORAGE?

The Object Storage Service uses virtual object containers to allow users to store and retrieve files and other data objects without a file system interface. The service's distributed architecture

supports horizontal scaling. Object redundancy is provided through software-based data replication. By supporting eventual consistency through asynchronous replication, it is well suited to data center deployments across different geographical areas.

Object storage uses the following technology:

- **Storage Replicas**

Used to maintain the state of objects in the case of outage. A minimum of three replicas is recommended.

- **Storage Zones**

Used to host replicas. Zones ensure that each replica of a given object can be stored separately. A zone might represent an individual disk drive or array, a server, all the servers in a rack, or even an entire data center.

- **Storage Regions**

A group of zones sharing a location. Regions can be groups of servers or server farms, usually located in the same geographical area. Regions have a separate API endpoint per Object Storage Service installation, which allows for discrete separation of services.

ARCHITECTURE OF OBJECT STORAGE

The following diagram shows the object proxy nodes split out from the service storage nodes which run the container and account services:

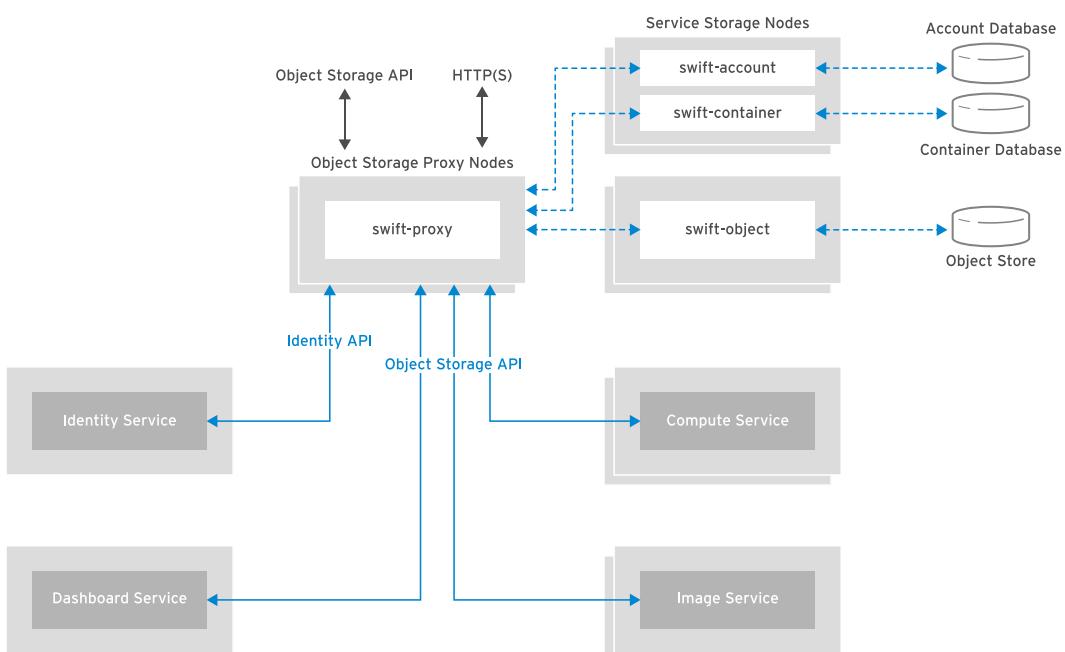


Figure 8.1: Swift Architecture

The Object Storage service is modular and is comprised of:

- **openstack-swift-proxy**

The proxy service uses the object ring to decide where to store newly uploaded objects. It updates the relevant container database to reflect the presence of a new object. If a newly uploaded object goes to a new container, the proxy service also updates the relevant account database to reflect the new container. The proxy service also directs GET requests to one of the nodes where a replica of the requested object is stored, either randomly or based on response time from the node. It provides access to the public Swift API, and is responsible for handling and

routing requests. The proxy server streams objects to users without spooling. Objects can also be served over HTTP. The proxy also possesses discovery mechanisms. Response to discovery requests include information about the cluster, and can be used by clients to determine which features are supported in the Swift cluster.

- **openstack-swift-account**

The account service maintains databases of all of the containers accessible by any given account. There is one database file for each account, and they are replicated across the cluster. Any account has access to a particular group of containers. An account maps to a project in the OpenStack identity service. The account service provides a list of what objects are in a specific container by referencing the container database.

- **openstack-swift-container**

The container service maintains databases of objects in containers. There is one database file for each container, and they are replicated across the cluster. Containers are defined when objects are put in them. Containers make finding objects faster by limiting object listings to specific container namespaces. The container service is responsible for listings of containers using the account database.

- **openstack-swift-object**

The object service is responsible for storing data objects in partitions on disk devices. Each partition is a directory, and each object is held in a subdirectory of its partition directory. An MD5 hash of the path to the object is used to identify the object itself. The service stores, retrieves, and deletes objects.

All of the services can be installed on each node or, alternatively, on dedicated machines. In addition, the following components are in place for proper operation:

- **Ring Files**

Ring files map names of stored entities to physical locations. They contain details of all storage devices, and are used to deduce where a piece of data is stored. One file is created for each object, account, and container server.

- **Supported File Systems**

The object service stores objects on file systems. XFS is the recommended file system, however ext4 is also supported. File systems must be mounted with xattr enabled to support metadata entries. The mount point is expected to be **/srv/node**.

- **Housekeeping Processes**

Replication ensures a consistent state in the event of outages or disk failures. Replication also ensures that files are removed when an object is deleted. *Auditors* run integrity checks on objects, containers, and accounts. If a corruption is discovered the replicator is notified and will replace the corrupt file from one of the other replicas.

SWIFT VS CEPH

Companies and organizations faced with storage solution decisions may feel challenged to choose between Swift and Ceph for their cloud-based deployments. Often these decisions are based on monetary factors or technology bias. However, the decision should be based on an understanding of the technologies and their features as they pertain to business needs. In some cases it may be logically reasonable to implement both technologies in the same deployment.

The following table lists just some of the features for Swift and Ceph. Explanations and use cases are presented following the feature table.

Storage Features

FEATURE	SWIFT	CEPH
Codebase	Python	Mostly C++
Consistency model	Eventual consistency	Strong consistency
Access	RESTful API	RESTful API
Object Storage	Y	Y
Block Storage	N	Y
File Based Storage	N	Y

Swift Feature Set Overview

- The advantage of the Python codebase is its extensible Web Server Gateway Interface (WSGI) pipeline to middleware plugins. For example, there are use cases for many different authentication systems which are available through middleware plugins that modify behavior to meet business needs.
- Eventual consistency presents a case for high availability in distributed computing that theoretically guarantees that, assuming no recent updates to a data item, requests for that item will return the last updated value.

An active use case for the eventual consistency model is the Internet Domain Name System (DNS), a proven system which demonstrates both high availability and scalability.

- Users can view, create, modify, and download objects and metadata using Object Storage API and standard HTTP calls using the Representational State Transfer (REST).
- The goal of Swift is to provide access to highly available and scalable object storage, and it does this well.

Ceph Feature Set Overview

- Objects are stored on Ceph Object Storage Devices (OSDs) in Ceph. Because the codebase in Ceph is mostly C++, clients can communicate directly with the OSDs, eliminating the need for a proxy service and therefore optimizing performance.
- The consistency model in Ceph is implemented by the RADOS Gateway, which provides strong consistency. Strong consistency means that new and updated objects are written to all disks before being visible to clients. Therefore, objects are only viewed from a single consistent state.

Use cases for Ceph include image storing, backup services, and file storing and sharing.

- Ceph Gateway, a RADOS Gateway, is an object storage interface. It provides applications with a RESTful gateway, and supports both Amazon S3 and Swift.
- The goal of Ceph is to provide more than just object storage. Ceph uses the same storage cluster for object, block, and file-based data storage.



REFERENCES

Further information is available in the Object Storage section of the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/storage_guide/

Further information is available in the Overview section of the *Architecture Guide* for Red Hat Ceph Storage at
https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html-single/architecture_guide/

DESCRIBING OBJECT STORAGE ARCHITECTURE

Choose the correct answers to the following questions:

► 1. Which two statements are true about object storage containers? (Choose two.)

- a. Containers are nested objects that organize file objects.
- b. Containers are storage compartments for objects.
- c. Nested pseudo-folders are allowed in containers.
- d. Containers are not designed for public access.

► 2. Which three statements are true about objects in the Object Storage service (swift)? (Choose three.)

- a. Retrieving an object is handled by a proxy service.
- b. Only a user with the admin role within a project can create objects for that particular account.
- c. Objects are stored as binary files.
- d. Objects are stored as ASCII text files.
- e. Objects are tracked by the object service in a database.
- f. For efficiency, metadata for an object is stored in the same file as the object.

► 3. Which three services have ring files created for them? (Choose three.)

- a. Proxy server
- b. Account server
- c. Object server
- d. Container server
- e. Host server

► 4. Which two storage features are common between Swift and Ceph storage solutions? (Choose two.)

- a. Consistency model
- b. Access
- c. Block storage
- d. File-based storage
- e. Object storage

DESCRIBING OBJECT STORAGE ARCHITECTURE

Choose the correct answers to the following questions:

► 1. Which two statements are true about object storage containers? (Choose two.)

- a. Containers are nested objects that organize file objects.
- b. Containers are storage compartments for objects.
- c. Nested pseudo-folders are allowed in containers.
- d. Containers are not designed for public access.

► 2. Which three statements are true about objects in the Object Storage service (swift)? (Choose three.)

- a. Retrieving an object is handled by a proxy service.
- b. Only a user with the admin role within a project can create objects for that particular account.
- c. Objects are stored as binary files.
- d. Objects are stored as ASCII text files.
- e. Objects are tracked by the object service in a database.
- f. For efficiency, metadata for an object is stored in the same file as the object.

► 3. Which three services have ring files created for them? (Choose three.)

- a. Proxy server
- b. Account server
- c. Object server
- d. Container server
- e. Host server

► 4. Which two storage features are common between Swift and Ceph storage solutions? (Choose two.)

- a. Consistency model
- b. Access
- c. Block storage
- d. File-based storage
- e. Object storage

MANAGING OBJECTS

OBJECTIVES

After completing this section, students should be able to:

- Create and delete containers.
- Create and delete folders.
- Upload and download objects.
- Use an object in an instance.

MANAGING CONTAINER OBJECTS USING THE DASHBOARD

Managing container objects involves several tasks that can be performed by individual project users or by administrators when a large number of data objects are stored for distribution such as a music or video download web site. This course focuses on the individual project user and the basic tasks for managing containers and container objects.

Managing OpenStack object storage containers and objects can be accomplished using several methods, including the dashboard, Object Storage API and Representational State Transfer (REST) web services, or the OpenStack unified CLI. This section focuses on the dashboard.

Creating a Container

- Containers are created by project users. Log in to the dashboard as a project user, then navigate to the Containers management interface at Project → Object Store → Containers.

The screenshot shows the 'Containers' page of the OpenStack dashboard. At the top, there's a navigation bar with 'Containers' underlined. Below it, a breadcrumb trail shows 'Project / Object Store / Containers'. The main area is titled 'Containers' and contains a button labeled '+ Container'. A search bar with the placeholder 'Click here for filters.' is present. To the right, a message says 'Select a container to browse.' Below the search bar, it says 'No items to display.'

Figure 8.2: Container Management Interface

- To create a container, click the +Container button to open the Create Container pop-up dialog interface, then enter a name in the Container Name dialog box and click the Submit button.



NOTE

Read the information in the right column of the Create Container dialog interface to gain additional knowledge about creating containers.

Create Container

Container Name *

 ✓

Container name must not contain "/".

Container Access

Public Not public

A Public Container will allow anyone with the Public URL to gain access to your objects in the container.

Cancel **Submit**

Figure 8.3: Create Container Dialog Interface

Creating a Pseudo-folder

- To create a pseudo-folder, click the container's name, then click the +Folder button.

The screenshot shows the Red Hat OpenStack Platform interface. The top navigation bar includes 'RED HAT OPENSTACK PLATFORM', 'Project', 'Admin', and 'Identity'. Below the navigation is a secondary bar with 'Compute', 'Network', 'Orchestration', and 'Object Store' tabs, with 'Object Store' being the active tab. Under 'Object Store', the 'Containers' section is selected. The main content area displays a list of containers, with 'demo-container' selected. A modal dialog box is open over the list, titled 'demo-container'. It contains a search bar ('Click here for filters.'), a table header ('Name' and 'Size'), and a message 'No items to display.' Below the table, it says 'Displaying 0 items'. At the bottom of the dialog are buttons for '+Folder' (highlighted in red) and a trash icon.

Figure 8.4: Create Pseudo-folder Interface

In the Create Folder In pop-up dialog interface, enter a name in the Folder Name dialog box, then click the +Create Folder button.

NOTE

Read the information in the right column of the Create Folder In dialog interface to gain additional knowledge about nesting folders.

Create Folder In: demo-container

Folder Name

Note: Delimiters ('/') are allowed in the folder name to create deep folders.

Cancel **+ Create Folder**

Figure 8.5: Create Folder Dialog Box

Uploading a File to a Folder

- To upload a file to a folder, click the container name, then click the name of the folder where you want to store the uploaded file. Click the upload icon located left of the +Folder button to open the Upload File To: interface.

The screenshot shows the 'Containers' section of a user interface. At the top, there's a breadcrumb navigation: Project / Object Store / Containers. Below it, a header says 'Containers' with a '+ Container' button. A search bar with placeholder text 'Click here for filters.' and a 'demo-container : demo-folder' label are visible. There are buttons for filtering by 'Name' or 'Size' and a 'Folder' button. A table lists one object: 'demo-upload-file.txt'. The table includes columns for 'Name' and 'Size'. A note at the bottom says 'No items to display.' and 'Displaying 0 items'.

Figure 8.6: Staging to Upload a File

Click Browse to locate the file object to upload, then click the Upload File button.

The screenshot shows the 'Upload File To:' dialog box. It has a title 'Upload File To: demo-container : demo-folder' and a close button. On the left, there's a 'File' section with a 'Browse...' button and the file path 'demo-upload-file.txt'. On the right, a note says: 'Note: Delimiters ('/') are allowed in the file name to place the new file into a folder that will be created when the file is uploaded (to any depth of folders.)'. Below this is a 'File Name' input field containing 'demo-upload-file.txt'. At the bottom right are 'Cancel' and 'Upload File' buttons.

Figure 8.7: Upload File To: Dialog

Download a File Object

- To download a file object, click the Download button. When the Opening *filename* dialog interface appears, click the Save File button to download the object.

The screenshot shows the 'Containers' section of a user interface. At the top, there's a breadcrumb navigation: Project / Object Store / Containers. Below it, a header says 'Containers' with a '+ Container' button. A search bar with placeholder text 'Click here for filters.' and a 'demo-container : demo-folder' label are visible. There are buttons for filtering by 'Name' or 'Size' and a 'Folder' button. A table lists two objects: 'demo-upload-file.txt'. The table includes columns for 'Name' and 'Size'. A 'Download' button is located to the right of the size column for each file. A note at the bottom says 'Displaying 1 item'.

Figure 8.8: Staging to Download a File

Delete a File Object

- To delete a file object, click the checkbox next to the name of the file to be deleted, then click the Trash Can icon to the right of the +Folder button.

The screenshot shows the 'Containers' section of the interface. On the left, there's a sidebar with a '+Container' button. The main area is titled 'demo-container : demo-folder'. It displays a table with one item: 'demo-upload-file.txt'. The table has columns for 'Name' and 'Size'. The 'Name' column shows 'demo-upload-file.txt' with a checked checkbox. The 'Size' column shows '46 bytes'. To the right of the table are buttons for 'Download' and a red trash can icon. At the bottom, it says 'Displaying 1 item'.

Figure 8.9: Delete Files Staging

In the Delete Files in *object-file-name* dialog interface, click the Delete button, then click OK to complete the deletion.

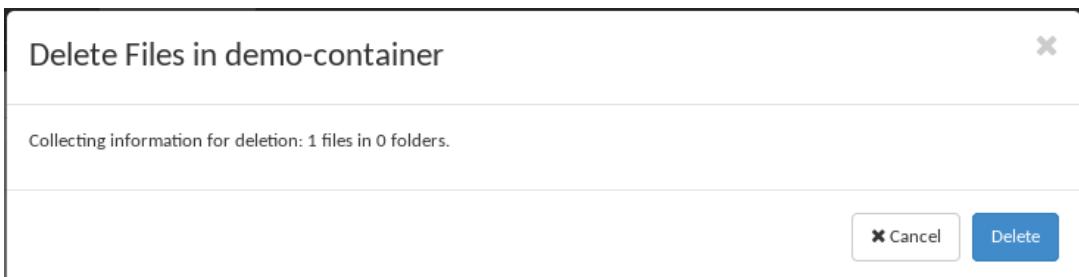


Figure 8.10: Delete Files Dialog

Delete a Pseudo-folder Object

- To delete a folder object, click the name of the container that holds the target folder, then in the menu of the folder to be deleted, click the Delete button.

The screenshot shows the 'Containers' section of the interface. On the left, there's a sidebar with a '+Container' button. The main area is titled 'demo-container'. It displays a table with one item: 'demo-folder'. The table has columns for 'Name' and 'Size'. The 'Name' column shows 'demo-folder' with a checked checkbox. The 'Size' column shows 'Folder'. To the right of the table is a red trash can icon with a 'Delete' label. At the bottom, it says 'Displaying 1 item'.

Figure 8.11: Delete Folder Staging

In the Delete Files in *container name* pop-up interface, click the Delete button, then click OK to complete the deletion.

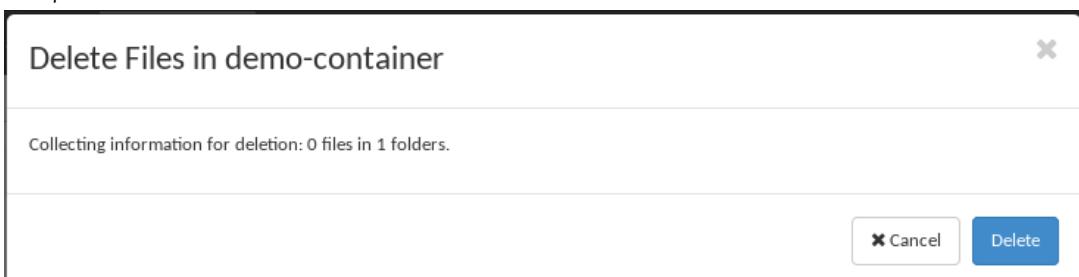


Figure 8.12: Delete Folder Dialog

Deleting a Container

- To delete a container, click the trash can icon next to the name of the container to be deleted.

The screenshot shows a user interface for managing containers. At the top, there's a navigation bar with 'Containers' selected. Below it is a breadcrumb trail: 'Project / Object Store / Containers'. The main area is titled 'Containers' and contains a button '+ Container'. A single container card is displayed with the following details:

demo-container	
Object Count:	0
Size:	0 bytes
Date Created:	Mar 14, 2017
<input type="checkbox"/> Public Access:	disabled

A small trash can icon is located in the top right corner of the container card.

Figure 8.13: Container Deletion Staging

In the Confirm Delete pop-up interface click Yes.

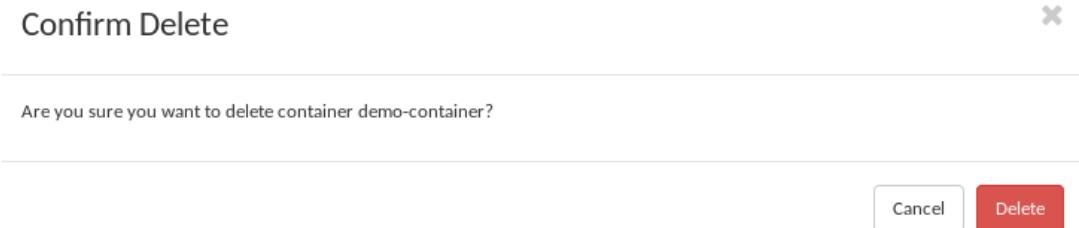


Figure 8.14: Confirm Deletion of Container

Managing Container Objects with the Dashboard

The following steps outline the process for creating, downloading, and deleting container objects using the dashboard.

- Open the dashboard in a web browser, then log in as a project user and navigate to Project → Object Store → Containers.
- Click the +Container button to open the Create Container pop-up dialog interface, then enter a name in the Container Name * dialog box. Click the +Create button.
- Click the container's name, then click the +Folder button to open the Create Folder In: pop-up dialog interface, then enter a name in the Folder Name dialog box. Click the +Create Folder button.
- To upload a file object to a folder, click the folder's name, then click the upload button located to the left of the +Folder button, then Browse the file system for the file object to upload. Click the Upload File button.
- To download a file object, click the Download button in the menu for the previously uploaded file. Click the Save File button.

- To delete a file object, click the checkbox next to the name of the file to be deleted, then click the Trash Can icon next to +Folder. In the Delete Files in *object-file-name* dialog interface, click the Delete button, then click OK to complete the deletion.
- To delete a folder object, click the name of the container where the folder is stored. In the menu of the folder to be deleted, click the Delete button. In the Delete Files in *container name* pop-up interface, click the Delete button, then click OK to complete the deletion.
- To delete a container, click the Trash Can icon next to the name of the container to be deleted. In the Confirm Delete pop-up interface, click Yes.

MANAGING CONTAINERS AND OBJECTS USING THE CLI

Managing containers includes creating, listing, and deleting containers. Managing pseudo-folders and container objects includes creating, listing, and deleting objects. Performing these tasks from the command-line using OpenStack unified CLI requires practice but can produce more expedient results as compared to navigating a graphical interface.

Creating, Listing, and Deleting a Container

To create a container, use the **openstack container create** command.

```
[user@demo ~]# openstack container create demo-container
+-----+-----+-----+
| account | container | x-trans-id |
+-----+-----+-----+
| AUTH_5ec4...ac99 | demo-container | tx85...2839 |
+-----+-----+-----+
```

To list all containers in a user's project, use the **openstack container list** command.

```
[user@demo ~]# openstack container list
+-----+
| Name |
+-----+
| demo-container |
+-----+
```

To delete an empty container, use the **openstack container delete** command. The delete option does not produce output to the console.

```
[user@demo ~]# openstack container delete demo-container
```

Creating, Listing, and Deleting Pseudo-folders and Objects

There are several ways to create pseudo-folders and file objects from the command-line.

Before beginning, create some objects to upload. Create a local folder named **/home/user/demo-folder**, then create a file in the folder named **demo-file.txt** with "random data" as its content.

```
[user@demo ~]# mkdir /home/user/demo-folder
[user@demo ~]# echo "random data" > /home/user/demo-folder/demo-file.txt
```

To create both objects at the same time in the Object Store, use the command, **openstack object create**. The command uploads the folder **demo-folder** to the container **demo-container** and upload the file **demo-file.txt** to the pseudo-folder stored in the container **demo-container**. It is important to note that the **demo-container** must exist for the following command to work.

```
[user@demo ~ (admin)]$ openstack object create demo-container \
demo-folder/demo-file.txt
+-----+-----+-----+
| object | container | etag |
+-----+-----+-----+
| demo-folder/demo-file.txt | demo-container | ea3292c15a25ca0d01eac9484180b446 |
+-----+-----+-----+
```

Use the following command to list all pseudo-folders and file objects within **demo-container**.

```
[user@demo ~ (admin)]$ openstack object list demo-container
+-----+
| Name |
+-----+
| demo-folder/demo-file.txt |
+-----+
```

Use the following command to delete an object from a pseudo-folder when there is only one object file in the pseudo-folder. The default behavior in this case is to delete both the file object and the pseudo-folder.

```
[user@demo ~ (admin)]$ openstack object delete demo-container demo-folder/demo-
file.txt
```

Use the following command to confirm that the file object, **demo-file.txt**, has been deleted. When the file object is not displayed in the output to the console, the deletion is confirmed.

```
[user@demo ~ (admin)]$ openstack object list demo-container
```

Use the following command to delete a single file object from a pseudo-folder when there are more than one object files in the pseudo-folder. The behavior in this case is to delete only the file object designated in the command.

Create a second file in the local directory **/home/user/demo-folder** named **/home/user/demo-folder/demo-file2.txt** with "random data2" as content.

```
[user@demo ~ (admin)]$ echo "random data2" > /home/user/demo-folder/demo-file2.txt
```

Upload (create) both files to pseudo-folder **demo-folder** in **demo-container**.

```
[user@demo ~ (admin)]$ openstack object create demo-container \
demo-folder/{demo-file.txt,demo-file2.txt}
+-----+-----+-----+
| object | container | etag |
+-----+-----+-----+
| demo-folder/demo-file.txt | demo-container | c673fdf123e68c5649a4cfb099ae1e06 |
| demo-folder/demo-file2.txt | demo-container | 753f467e07e1b39c47600a4c5ece8be6 |
+-----+-----+-----+
```

Use the following command to delete the file object, **demo-file.txt**, from the pseudo-folder **demo-folder**. The delete option does not produce output to the console.

```
[user@demo ~]# openstack object delete demo-container \
demo-folder/demo-file.txt
```

Confirm that the file object **demo-file.txt** has been deleted and the file object **demo-file2.txt** remains.

```
[user@demo ~]# openstack object list demo-container
+-----+
| Name           |
+-----+
| demo-folder/demo-file2.txt |
+-----+
```

NOTE

By default, a user that has been assigned the admin role has full access to all containers and objects in an account. If an external user requires access to the container objects, an access control list (ACL) can be set at the container level and support lists for read and write access. For example, to allow read and list access of objects in a container named **web** for external users use the **swift post** command:

```
[user@demo ~]# swift post -r '.r:*,.rlistings' web
```

Managing Container Objects with the Command-line Interface

The following steps outline the process for creating, downloading, and deleting container objects using the command-line interface.

1. Source the identity environment file for a user.
2. Create a container using the **openstack container create** command.
3. Upload a file object to a new folder within an existing container using the **openstack object create** command.
4. Download a file object from the Object Store to a local directory, using the **openstack object save** command.
5. Delete a file object from the container folder using the **openstack object delete** command.
6. Folders must be deleted from a container before the container can be deleted. Delete an empty container using the **openstack container delete** command.

REFERENCES

Further information is available in the Object Storage section of the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/storage_guide/

MANAGING OBJECTS

In this exercise, you will create and delete object containers, folder objects, and file objects.

OUTCOMES

You should be able to:

- Create, view, and delete object containers.
- Create, list, download, and delete container objects.

Confirm the **workstation** and overcloud's virtual machines are started.

Log in to **workstation** as **student** user with **student** as the password. From **workstation**, run **lab manage-objects setup**, which ensures that the OpenStack services are running and the resources created previously are available to complete this exercise.

```
[student@workstation ~]$ lab manage-objects setup
```

- ▶ 1. Log in to the dashboard as **developer1**.
 - 1.1. On **workstation** open Firefox and browse to `http://dashboard.overcloud.example.com`.
 - 1.2. Login to the dashboard with user name **developer1** and password **redhat**.
- ▶ 2. Create a container named **developer1-container1**.
 - 2.1. If Containers is not the active page element, navigate to the left side of the menu bar and click the dropdown menu named Object Store, then select Containers to activate the interface for managing containers.
 - 2.2. Click the +Container button. In the Create Container dialog pop-up, navigate to the Container Name text box and enter **developer1-container1**. Make sure that the Container Access > Public selection is **not checked**.
Click the Submit button.
- ▶ 3. Create a folder in **developer1-container1** container named **folder1**.
 - 3.1. On the left side of the interface, click the container named **developer1-container1** to activate the management interface for the container.
 - 3.2. On the right side of the Containers management interface, click the +Folder button, then in the Create Folder In: **developer1-container1** dialog pop-up, navigate to the Folder Name text field and enter **folder1**.
Click the +Create Folder button.
- ▶ 4. On **workstation**, in **student**'s home directory, create a file named **developer1-helloworld.txt** that later will be uploaded to the **folder1** folder in the **developer1-container** container.

```
[student@workstation ~]$ mkdir /home/student/folder1
[student@workstation ~]$ echo 'Hello World!' > \
/home/student/folder1/developer1-helloworld.txt
```

- ▶ 5. In the dashboard, upload the **developer1-helloworld.txt** file into the folder named **folder1** located in the object store's container named **developer1-container1**.
 - 5.1. From **workstation**, if necessary, open Firefox and log in to the dashboard as **developer1** and navigate to Object Store → Containers. Click the developer1-container1 container name to activate its management interface.
 - 5.2. Click the folder1 folder name in the *Folder* subsection. You should see developer1-container1 : **folder1** above the *folder* section of the interface.
 - 5.3. On the right side of the interface, click the Upload button located to the left of the +Folder button.
 - 5.4. Navigate to Upload File To: developer1-container1 : folder1 pop-up dialog, then under File use the Browse button to locate the file **developer1-helloworld.txt** in the directory **/home/student/folder1**. Select **developer1-helloworld.txt**, click Open, and then click the Upload File button to upload the file.
- ▶ 6. On **workstation** use the command-line interface to delete the **developer1-helloworld.txt** file, the **folder1** folder, and **developer1-container1** container.
 - 6.1. In **student**'s home directory, source the **/home/student/developer1-finance-rc** environment file to export the **developer1** user identity.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$
```

- 6.2. Verify available containers in the object store.

```
[student@workstation ~(developer1-finance)]$ openstack container list
+-----+
| Name          |
+-----+
| developer1-container1 |
+-----+
```

- 6.3. Verify available file objects in the **developer1-container1** container.

```
[student@workstation ~(developer1-finance)]$ openstack object list \
developer1-container1
+-----+
| Name          |
+-----+
| folder1/      |
| folder1/developer1-helloworld.txt |
+-----+
```

- 6.4. Delete the file object **developer1-helloworld.txt**.

```
[student@workstation ~(developer1-finance)]$ openstack object delete \
developer1-container1/folder1 \
```

developer1-helloworld.txt

Verify that the file object **developer1-helloworld.txt** has been deleted.

```
[student@workstation ~-(developer1-finance)]$ openstack object list \
developer1-container1
+-----+
| Name      |
+-----+
| folder1/   |
+-----+
```

6.5. Delete the folder object **folder1**.

```
[student@workstation ~-(developer1-finance)]$ openstack object delete \
developer1-container1 \
folder1/
```

Verify that the folder object **folder1** has been deleted. No objects should be listed in the output.

```
[student@workstation ~-(developer1-finance)]$ openstack object list \
developer1-container1
```

6.6. Delete the container **developer1-container1**.

```
[student@workstation ~-(developer1-finance)]$ openstack container delete \
developer1-container1
```

Verify that the container **developer1-container1** has been deleted.

```
[student@workstation ~-(developer1-finance)]$ openstack container list
```

- 7. On **workstation**, use the command-line interface to create a container named **developer1-container2**, a folder object named **folder2**, and a file object named **developer1-data.txt**.

7.1. Create a container named **developer1-container2**.

```
[student@workstation ~-(developer1-finance)]$ openstack container create \
developer1-container2
+-----+
| account      | container          | x-trans-id    |
+-----+-----+-----+
| AUTH_35f2...6c07 | developer1-container2 | tx9935...6882 |
+-----+-----+-----+
```

7.2. For the purpose of this exercise, create local copies of the directory **folder2** and the file **developer1-data.txt** in **student**'s home directory for uploading to the object store.

```
[student@workstation ~-(developer1-finance)]$ mkdir /home/student/folder2
[student@workstation ~-(developer1-finance)]$ echo "command-line interface \
```

```
directory environment" > /home/student/folder2/developer1-data.txt
```

- 7.3. Create a folder object named **folder2** and a file object named **developer1-data.txt** within the container named **developer1-container2**.

```
[student@workstation ~-(developer1-finance)]$ openstack object create \
developer1-container2 folder2/developer1-data.txt
+-----+-----+-----+
| object | container | etag |
+-----+-----+-----+
| folder2/developer1-data.txt | developer1-container2 | b40b...e153 |
+-----+-----+-----+
```

- 8. On **workstation**, download the file object **hosts** from the object store to **student's / home/student/Downloads** directory.

- 8.1. Verify available containers in the object store.

```
[student@workstation ~-(developer1-finance)]$ openstack container list
+-----+
| Name |
+-----+
| developer1-container2 |
+-----+
```

- 8.2. Verify available file objects in the **developer1-container2** container.

```
[student@workstation ~-(developer1-finance)]$ openstack object list \
developer1-container2
+-----+
| Name |
+-----+
| folder2/developer1-data.txt |
+-----+
```

- 8.3. Save the object file **developer1-data.txt** from the object store to **/home/student/Downloads** as **data.txt**.

```
[student@workstation ~-(developer1-finance)]$ openstack object save \
developer1-container2/folder2 \
developer1-data.txt \
--file /home/student/Downloads/data.txt
```

- 8.4. Verify that the file object **developer1-data.txt** successfully downloaded to **/home/student/Downloads/data.txt**.

```
[student@workstation ~-(developer1-finance)]$ cat \
/home/student/Downloads/data.txt
command-line interface directory environment
```

Cleanup

From **workstation**, run the **lab manage-objects cleanup** script to remove extraneous artifacts.

```
[student@workstation ~]$ lab manage-objects cleanup
```

This concludes the guided exercise.

MANAGING OBJECT STORAGE

PERFORMANCE CHECKLIST

In this lab, you will create a container, add a folder object to the container, and add a file object in the folder.

OUTCOMES

You should be able to:

- Create containers.
- Add folder and file objects to a container.
- Download container objects.

Confirm the **workstation** and overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as a password. From **workstation**, run **lab manage-objects-lab setup**, which ensures that the resources created previously are available to complete this lab.

```
[student@workstation ~]$ lab manage-objects-lab setup
```

1. From workstation, as the **operator1** user, create a container named **production-container1**, and then verify that the container exists.
2. Create two objects, a folder object named **html** and a text file object within it named **index.html**. In the **index.html** file, include the single line **Hello world!**. Once created, upload the objects to the **production-container1** container.
3. Confirm that the objects uploaded successfully by listing the objects in the **production-container1** container. Save the **index.html** object to the **/var/www/html** directory.

Evaluation

From **workstation**, use the **lab manage-objects-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab manage-objects-lab grade
```

Cleanup

From **workstation**, run the **lab manage-objects-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab manage-objects-lab cleanup
```

This concludes the lab.

MANAGING OBJECT STORAGE

PERFORMANCE CHECKLIST

In this lab, you will create a container, add a folder object to the container, and add a file object in the folder.

OUTCOMES

You should be able to:

- Create containers.
- Add folder and file objects to a container.
- Download container objects.

Confirm the **workstation** and overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as a password. From **workstation**, run **lab manage-objects-lab setup**, which ensures that the resources created previously are available to complete this lab.

```
[student@workstation ~]$ lab manage-objects-lab setup
```

1. From workstation, as the **operator1** user, create a container named **production-container1**, and then verify that the container exists.

1.1. Open a terminal and source the **operator1** user's environment file.

```
[student@workstation ~]$ source ~/operator1-production-rc
```

1.2. Create a container named **production-container1**.

```
[student@workstation ~(operator1-production)]$ openstack container create \
production-container1
+-----+-----+-----+
| account | container | x-trans-id |
+-----+-----+-----+
| AUTH_3b33...246d | production-container1 | txbe...bb51 |
+-----+-----+-----+
```

1.3. Verify that the **production-container1** container was successfully created by listing the existing containers.

```
[student@workstation ~(operator1-production)]$ openstack container list
+-----+
| Name |
+-----+
| production-container1 |
```

2. Create two objects, a folder object named **html** and a text file object within it named **index.html**. In the **index.html** file, include the single line **Hello world!**. Once created, upload the objects to the **production-container1** container.

2.1. Create a directory object in student's home directory named **/home/student/html**.

```
[student@workstation ~-(operator1-production)]$ mkdir /home/student/html
```

2.2. Add a text file object to the directory named **/home/student/html/index.html**.

Populate the object file **index.html** with a single line that reads **Hello world!**.

```
[student@workstation ~-(operator1-production)]$ echo 'Hello world!' > \
/home/student/html/index.html
```

2.3. Upload the combined objects **html/index.html** to the **production-container1** container.

```
[student@workstation ~-(operator1-production)]$ openstack object create \
production-container1 html/index.html
+-----+-----+-----+
| object | container | etag |
+-----+-----+-----+
| html/index.html | production-container1 | 8ddd...9858 |
+-----+-----+-----+
```

3. Confirm that the objects uploaded successfully by listing the objects in the **production-container1** container. Save the **index.html** object to the **/var/www/html** directory.

3.1. Use the **openstack object list** command to identify the objects stored in the **production-container1** container.

```
[student@workstation ~-(operator1-production)]$ openstack object list \
production-container1
+-----+
| Name |
+-----+
| html/index.html |
+-----+
```

3.2. Log in to **workstation** as root. The password is **student**.

```
[student@workstation (operator1-production)]$ sudo -i
[sudo] password for student: student
[root@workstation ~]#
```

3.3. Source the **/home/student/operator1-production-rc** environment file. Using the **openstack object save** command download the **index.html** object to the **/var/www/html** directory. Check that the file has been downloaded.

```
[root@workstation ~]# source /home/student/operator1-production-rc
[root@workstation (operator1-production)]# openstack object save \
--file /var/www/html/index.html production-container1 \
html/index.html
[root@workstation (operator1-production)]# ll /var/www/html
```

```
total 4  
-rw-rw-r--. 1 root root 13 Jun 15 16:18 index.html  
[root@workstation (operator1-production)]# cat /var/www/html/index.html  
Hello World!
```

Evaluation

From **workstation**, use the **lab manage-objects-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab manage-objects-lab grade
```

Cleanup

From **workstation**, run the **lab manage-objects-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab manage-objects-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- A virtual container in object storage is a storage compartment for objects.
- An object is the actual data stored as a binary file along with its metadata, which is stored in the file's extended attributes.
- Pseudo-folders can be created in the virtual containers to better organize object storage.
- OpenStack object storage provides a storage solution of virtual containers that allow users to store and retrieve objects.
- The OpenStack object storage proxy server uses object ring files to determine where to create new objects, or where to find an existing object to update or retrieve. The proxy server is the communication mechanism that processes requests for the OpenStack object storage API.
- Swift and Ceph both offer storage solutions for cloud-based deployments. Each provide features that are common to one another, such as a RESTful API. Individually, they have their own distinct feature set that allows them to stand on their own as a storage solution. Additionally, they could be deployed together when their combined features address particular business needs.

PREPARING TO DEPLOY AN INSTANCE WITH PUBLIC ACCESS

GOAL

Manage external networks and security in preparation for launching an instance with public access.

OBJECTIVES

- Manage external networks
- Manage OpenStack routers
- Manage floating IP addresses
- Manage SSH key pairs and security groups

SECTIONS

- Managing External Networks (and Guided Exercise)
- Preparing OpenStack Routers (and Guided Exercise)
- Maintaining Floating IP Addresses (and Guided Exercise)
- Implementing Security (and Guided Exercise)

LAB

- Preparing to Deploy an Instance with Public Access

MANAGING EXTERNAL NETWORKS

OBJECTIVES

After completing this section, students should be able to:

- Create an external network and subnet
- View status of NetworkManager and network
- View Ethernet configuration
- Show OVS bridges

OPENSTACK NETWORKING

The OpenStack networking Service (neutron) is the networking framework for OpenStack. The OpenStack networking Service provides a set of services which aim to provide an interface for defining network connectivity and network addressing in OpenStack environments. The OpenStack networking service uses a plug-in based architecture providing the required flexibility for different networking technologies and topologies.

Deploy complex network topologies by creating and configuring networks, subnets, virtual devices, and network routing. The OpenStack networking service supports multiple private networks. Moreover, projects can deploy their own IP addressing scheme, even if it overlaps with those used by other projects.

The Neutron networking service uses plug-ins to provide a pluggable API back end. Plug-ins can be written for a variety of vendor technologies. Neutron networking service plug-ins support VLANs, as well as other technologies, such as L2-in-L3 tunneling and OpenFlow.

Subnets in OpenStack

OpenStack instances are created on tenant networks in projects. Each tenant network has a subnet which controls network connectivity. Each project can have one or more tenant networks, which enables network isolation between different systems using the same physical network.

Project Networks

Projects in OpenStack have access to a variety of network types, such as VLAN segregation, where every network uses a unique VLAN identifier. Projects can also use Virtual Extensible LAN (VXLAN). VXLAN is a network virtualization technology that solves the scalability problems associated with large cloud computing deployments. It increases scalability up to 16 million logical networks and allows for layer 2 adjacency across IP networks. VXLANs are designed to encapsulate Layer-2 networks and tunnel them over Layer-3 networks.

External Networks

External networking allows an instance to access a network outside its private range, such as the Internet. This type of network uses network address translation (*NAT*) by default. For external traffic to reach the instance, administrators must create the security rules per traffic type, such as SSH, or ICMP. Moreover, a floating IP needs to be associated to the instance. In OpenStack, external networks are special types of networks which can be shared between multiple projects. External routing is managed by the Neutron L3 agent, which provides Layer-3 routing and NAT forwarding.

Provider Networks

Instances are attached to existing physical networks via the provider network. As the name suggests, provider networks are directly mapped to pre-existing networks in physical infrastructures. This allows external systems and OpenStack instances to share the same layer-2 network.

Network Namespaces

The kernel provided by the Red Hat Enterprise Linux 7 repositories supports network namespaces. Network namespaces cover network access with a layer of abstraction by virtualizing access to the network resources, such as ports and interfaces. Both the `ip` command and the `iproute` command can be used to query the resources present in network namespaces.

NETWORKING WORKFLOW ON INSTANCES CREATION

The following describes a basic workflow when a new instance is created in an environment which uses Open vSwitch as the network back end.

- When an instance is created, the compute service queries the networking service API to find the set of required resources, such as the network port. The networking service then places a request for an IP address to be assigned. Such requests are managed by the networking DHCP service, which uses the `dnsmasq` daemon. The `dnsmasq` daemon returns an available IP address from the subnet's defined range.
- After an IP address is assigned to the new instance, the OpenStack networking service requests Open vSwitch to set up the required configuration, such as the network ports, to connect the instance to the subnet.
- Optionally, a router is created in the network environment to provide external access to the instance. OpenStack creates a set of Netfilter rules to configure the network routing.

Managing Networks

Network creation in OpenStack enables instances to receive proper addressing via DHCP in order to be reachable. The integration with physical networks enables systems outside of the OpenStack infrastructure to communicate with an instance. OpenStack networks can manage multiple subnets, enabling different instances to be hosted in the same system in complete isolation.

Managing External Networks Using the Dashboard

The following steps outline the process for managing external networks in Red Hat OpenStack Platform using the Dashboard.

1. Log in to the Dashboard as an administrative user and navigate to Network → Networks.
2. Click + Create Network to create a new network. Give the network a name and click Next.
3. In the Subnet tab, enter a subnet name as well as a subnet range using the CIDR notation and click Next.
4. Choose whether or not to enable the DHCP service for the network and enter a range to use as the allocation pool.

Managing External Networks Using the Command-Line Interface

The following steps outline the process for managing external networks in Red Hat OpenStack Platform using the `openstack` command-line interface.

1. Source the identity environment of the administrative user for the project.

2. Create a network using the **openstack network create** command. Create the network as external and shared. Include the name of the physical network provider and the network provider type.
3. Source the identity environment of a normal user.
4. Create a subnet for the network using the **openstack subnet create** command. Ensure that the subnet belongs to a range which is externally accessible.
5. Use the **systemctl status NetworkManager** command to view the status of the NetworkManager service and the **systemctl status network** command to view the status of the traditional network service.
6. Use the **ovs-vsctl** command to review the bridges as well as their ports.
7. The network configuration file **/etc/sysconfig/network-scripts/ifcfg-br-ex** contains the network definition for the Open vSwitch bridge. The network configuration file **/etc/sysconfig/network-scripts/ifcfg-eth0** contains the network definition for the network port enslaved to the Open vSwitch bridge.



REFERENCES

Further information is available in the *Networking Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

Further information is available in the *RFC1122* at

<https://tools.ietf.org/html/rfc1122>

MANAGING EXTERNAL NETWORKS

In this exercise, you will use the dashboard and the OpenStack unified CLI to create and configure an external network. You will use the command-line interface to review how the network is implemented.

OUTCOMES

You should be able to:

- Create an external network using the dashboard.
- Create a network using the **openstack** command-line interface.
- Review the implementation of the network topology in the virtual machine **controller0**.

Log in to workstation as **student** using **student** as the password. Run the **lab prep-deploy-ext setup** command, which ensures that the required users and projects are present and the identity environment files exist.

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

- ▶ 1. Create an external network using the dashboard. Name the network **provider-datacentre**. Enter **finance** as the project, **Flat** as the network type, and **datacentre** as the physical network. Mark the network as shared and external.
- 1.1. From **workstation**, open a web browser and navigate to `http://dashboard.overcloud.example.com`. Log in using **architect1** as the username, and **redhat** as the password.
 - 1.2. Within the Admin tab, navigate to Network → Networks.
 - 1.3. Click + Create Network to create a new network. Enter **provider-datacentre** as the Network Name. Select **finance** as the Project and Flat as the Provider Network Type. Enter **datacentre** as the Physical Network. Check Shared and External Network.
 - 1.4. Click on the Next button to create the subnet. Enter **provider-subnet-172.25.250** as the Subnet Name. Enter **172.25.250.0/24** as the Network Address. Enter **172.25.250.254** as the Gateway IP. Click on the Next button to make the final settings for the subnet.
 - 1.5. Disable **DHCP** by unclicking the Enable DHCP check box. In the Allocation Pools enter the following IP addresses **172.25.250.101, 172.25.250.189**. Enter **172.25.250.254** as the DNS Name Servers. Click on Create.
- ▶ 2. From **workstation**, source the identity environment of the **architect1** user and delete the **provider-datacentre** network and the **provider-subnet-172.25.250** subnet.
- 2.1. Open a terminal and source the identity environment for the **architect1** user, located in the `/home/student/` directory.

```
[student@workstation ~]$ source ~/architect1-finance-rc
```

```
[student@workstation ~ (architect1-finance)]$
```

2.2. Delete the subnet **provider-subnet-172.25.250**.

```
[student@workstation ~ (architect1-finance)]$ openstack subnet delete \
provider-subnet-172.25.250
```

2.3. Delete the network **provider-datacentre**.

```
[student@workstation ~ (architect1-finance)]$ openstack network delete \
provider-datacentre
```

- 3. Create an external network using the OpenStack unified CLI. Name the network **provider-datacentre**. Set the network type to **flat**, the physical network to **datacentre**. Set the network as shared and external.

3.1. As the **architect1** user, create the network **provider-datacentre**.

```
[student@workstation ~ (architect1-finance)]$ openstack network create \
--external \
--share \
--provider-network-type flat \
--provider-physical-network datacentre \
provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T13:44:28Z |
| description | |
| dns_domain | None |
| id | 29840e5f-3c29-4937-b6f8-237b51c21899 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1500 |
| name | provider-datacentre |
| port_security_enabled | True |
| project_id | e6b7f37fadf943e1aae06a5c2e55d646 |
| provider:network_type | flat |
| provider:physical_network | datacentre |
| provider:segmentation_id | None |
| qos_policy_id | None |
| revision_number | 6 |
| router:external | External |
| segments | None |
| shared | True |
| status | ACTIVE |
| subnets | |
| tags | |
| updated_at | 2018-06-21T13:44:28Z |
+-----+-----+
```

- 4. As the **developer1** user, create the subnet **provider-subnet-172.25.250** for the **provider-datacentre** network. Give the subnet a range of **172.25.250.101** to **172.25.250.189**. Disable DHCP services for the subnet and use **172.25.250.254** as the gateway as well as the DNS name server.

4.1. Source the identity environment for the **developer1** user.

```
[student@workstation ~-(architect1-finance)]$ source ~/developer1-finance-rc  
[student@workstation ~-(developer1-finance)]$
```

4.2. Create the subnet **provider-subnet-172.25.250** in the **provider-datacentre** network.

```
[student@workstation ~-(developer1-finance)]$ openstack subnet create \  
--subnet-range 172.25.250.0/24 \  
--no-dhcp \  
--gateway 172.25.250.254 \  
--dns-nameserver 172.25.250.254 \  
--allocation-pool start=172.25.250.101,end=172.25.250.189 \  
--network provider-datacentre \  
provider-subnet-172.25.250  
+-----+  
| Field | Value |  
+-----+  
| allocation_pools | 172.25.250.101-172.25.250.189 |  
| cidr | 172.25.250.0/24 |  
| created_at | 2018-06-21T13:51:02Z |  
| description | |  
| dns_nameservers | 172.25.250.254 |  
| enable_dhcp | False |  
| gateway_ip | 172.25.250.254 |  
| host_routes | |  
| id | 1f5e3ad7-25b0-4e49-ad7e-d6673e2882c9 |  
| ip_version | 4 |  
| ipv6_address_mode | None |  
| ipv6_ra_mode | None |  
| name | provider-subnet-172.25.250 |  
| network_id | 29840e5f-3c29-4937-b6f8-237b51c21899 |  
| project_id | e6b7f37fadf943e1aae06a5c2e55d646 |  
| revision_number | 0 |  
| segment_id | None |  
| service_types | |  
| subnetpool_id | None |  
| tags | |  
| updated_at | 2018-06-21T13:51:02Z |  
+-----+
```

- 5. Review the implementation of the networks on the **controller0** node.

5.1. Use the **ssh** command to connect to the **controller0** node.

```
[student@workstation ~-(developer1-finance)]$ ssh heat-admin@controller0
```

```
[heat-admin@controller0 ~]$
```

- 5.2. Confirm that even though NetworkManager is running, it does not control the network interfaces managed by OpenStack. First, ensure that the **NetworkManager** service is running and enabled.

```
[heat-admin@controller0 ~]$ 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 2018-07-03 07:45:32 UTC; 1h 16min ago
       Docs: man:NetworkManager(8)
   Main PID: 861 (NetworkManager)
     Tasks: 3
    Memory: 7.1M
      CGroup: /system.slice/NetworkManager.service
              └─861 /usr/sbin/NetworkManager --no-daemon
...output omitted...
```

- 5.3. Display the value of the **NM_CONTROLLED** parameter in the **/etc/sysconfig/network-scripts/ifcfg-*** files to verify that none of the interfaces are managed by NetworkManager.

```
[heat-admin@controller0 ~]$ cd /etc/sysconfig/network-scripts
[heat-admin@controller0 network-scripts]$ grep NM_CONTROLLED ifcfg-*
ifcfg-br-ex:NM_CONTROLLED=no
ifcfg-br-int:NM_CONTROLLED=no
ifcfg-br-trunk:NM_CONTROLLED=no
ifcfg-eth0:NM_CONTROLLED=no
ifcfg-eth1:NM_CONTROLLED=no
ifcfg-eth2:NM_CONTROLLED=no
...output omitted...
[heat-admin@controller0 network-scripts]$ cd
[heat-admin@controller0 ~]$
```

- 5.4. Use the **ovs-vsctl** command to review the network configuration. List all the bridges and the ports. The **br-ex** bridge contains three ports: **eth2**, which is the physical device, **phy-br-ex** and **br-ex**, which are the virtual interfaces.

```
[heat-admin@controller0 ~]$ sudo ovs-vsctl show
...output omitted...
Bridge br-ex
  Controller "tcp:127.0.0.1:6633"
    is_connected: true
    fail_mode: secure
    Port "eth2"
      Interface "eth2"
    Port phy-br-ex
      Interface phy-br-ex
        type: patch
        options: {peer=int-br-ex}
    Port br-ex
      Interface br-ex
        type: internal
Bridge br-int
```

```
Controller "tcp:127.0.0.1:6633"
    is_connected: true
    fail_mode: secure
    Port patch-tun
        Interface patch-tun
            type: patch
            options: {peer=patch-int}
    Port int-br-ex
        Interface int-br-ex
            type: patch
            options: {peer=phy-br-ex}
...output omitted...
```

- 5.5. Run the **ip** command to check the allocation of the IP addresses. Locate the **br-ex** and **eth2** devices. The **br-ex** device has the IP address of **172.25.250.1**.

```
[heat-admin@controller0 ~]$ ip a
...output omitted...
2: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-system state UP qlen 1000
    link/ether 52:54:00:00:fa:0b brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5054:ff:fe00:fa0b/64 scope link
        valid_lft forever preferred_lft forever
...output omitted...
5: br-ex: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN qlen 1000
    link/ether 22:b3:01:d9:81:49 brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.1/24 brd 172.25.250.255 scope global br-ex
        valid_lft forever preferred_lft forever
    inet6 fe80::20b3:1ff:fed9:8149/64 scope link
        valid_lft forever preferred_lft forever
```

- 5.6. Review the network configuration file for the **eth2** device. **eth2** is set as an Open vSwitch port for the **br-ex** network device.

```
[heat-admin@controller0 ~]$ cd /etc/sysconfig/network-scripts
[heat-admin@controller0 network-scripts]$ cat ifcfg-eth2
DEVICE=eth2
ONBOOT=yes
TYPE=OVSPort
DEVICETYPE=ovs
OVS_BRIDGE=br-ex
```

- 5.7. Review the network configuration file for the **br-ex** device. **br-ex** is declared as an Open vSwitch bridge and contains the IPv4 network configuration that ensures external routing of the OpenStack networks.

```
[heat-admin@controller0 network-scripts]$ cat ifcfg-br-ex
DEVICE=br-ex
HOTPLUG=no
ONBOOT=yes
NM_CONTROLLED=no
PEERDNS=no
DEVICETYPE=ovs
TYPE=OVSBridge
```

```
BOOTPROTO=static
IPADDR=172.25.250.1
NETMASK=255.255.255.0
OVS_EXTRA="set bridge br-ex other-config:hwaddr=52:54:00:02:fa:01 -- set bridge
br-ex fail_mode=standalone -- del-controller br-ex"
```

5.8. Exit from the **controller0** virtual machine.

```
[heat-admin@controller0 network-scripts]$ exit
[student@workstation ~ (developer1-finance)]$
```

Cleanup

From **workstation**, run the **lab prep-deploy-ext** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab prep-deploy-ext cleanup
```

This concludes the guided exercise.

PREPARING OPENSTACK ROUTERS

OBJECTIVES

After completing this section, students should be able to:

- Discuss routers and their implementation in OpenStack.
- Manage routers in OpenStack.

INTRODUCTION TO ROUTERS

Whether using IPv4 or IPv6, network traffic needs to move from host to host and network to network. Each host has a *routing table*, which instructs how to route the traffic for a particular network. Routing table entries list a destination network, the interface to send the traffic out, and the IP address of any intermediate router that is required to relay the message to its final destination. The routing table entry which matches the destination of the network traffic is used to route it. If two entries match a network traffic, the one with the longest prefix is used for handling the routing.

If the network traffic does not match a more specific route, the routing table usually has an entry for a *default route* to the entire IPv4 Internet, indicated as **0.0.0.0/0**. This default route points to a *router* on a reachable subnet, that is, on a subnet that has a more specific route in the host's routing table. If a router receives traffic that is not addressed to it, instead of ignoring it, it forwards the traffic based on its own routing table. This may send the traffic directly to the destination host, if the router happens to be on the destination's subnet, or it may be forwarded on to another router. This process of forwarding continues until the traffic reaches its final destination. The **ip** command supports the management of routes, and can be used to display the routes present in the system as shown in the following example.

```
[root@demo ~(admin)]# ip route show
default ❶ via 172.25.250.254 ❷ dev br-ex ❸
172.24.250.0/24 ❹ dev eth1 ❺ proto kernel scope link src 172.24.250.11 ❻
172.25.250.0/24 dev br-ex proto kernel scope link src 172.25.250.11
```

- ❶ This entry handles the routing for the network traffic that does not match a specific route.
- ❷ This entry indicates the IP address to which the traffic is forwarded to for this entry.
- ❸ This entry indicates the device through which the traffic is forwarded.
- ❹ This entry handles the routing for the network traffic in the **172.24.250.0/24** network range.
- ❺ This entry indicates the network device through which the network is handled for the range.
- ❻ This entry indicates the IP address of the device.

IMPLEMENTATION OF ROUTERS IN OPENSTACK

For the instances to communicate with any external subnet, a router must be deployed. The Red Hat OpenStack Platform provides such routing capabilities via Software-defined Networking (*SDN*). SDN-based virtual routers are similar to physical routers, and they require one subnet per interface. The traffic received by the router uses the router's default gateway as the next *hop*, which is one portion of the path between the source and the destination. The default gateway uses

a virtual bridge to route the traffic to an external network. In OpenStack, routers have as many interfaces as the subnets for which they route traffic.

When routers are created, the Neutron service creates a set of new routing rules in the network namespace of the virtual router. Such routes provide isolated routing from the default network stack. The following example shows how the route for the **192.168.3.0/24** network uses a specific network device, which is only present in the network namespace.

```
[root@demo ~(admin)]# ip netns exec qrouter-UUID ip a
... output omitted ...

13: qr-a669ab8c-1d: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN qlen 1000
    link/ether fa:16:3e:67:22:60 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.1/24 brd 192.168.3.255 scope global qr-a669ab8c-1d
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe67:2260/64 scope link
        valid_lft forever preferred_lft forever

... output omitted ...

[root@demo ~(admin)]# ip netns exec qrouter-UUID route
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
default         workstation.lab 0.0.0.0      UG    0      0      0
    qg-e4929eb0-cd
172.25.250.0   0.0.0.0       255.255.255.0 U      0      0      0
    qg-e4929eb0-cd
192.168.3.0    0.0.0.0       255.255.255.0 U      0      0      0
    qr-a669ab8c-1d
```

CREATING ROUTERS

The following describes a basic workflow when a new router is created.

- When a router is created for a project, the Neutron networking service creates a new network namespace, with a name of **qrouter-UUID**.

```
[root@@demo ~]# ip netns list
qrouter-f0785aec-f535-46f2-b93b-0acff4e951ef
```

- The Neutron networking service creates the required ports in its database. The entry identifies the subnets to connect the router to, the physical servers, as well as the security groups.

```
[root@demo ~(admin)]# openstack port list -c ID
+-----+
| ID
+-----+
| ec95a76e-3471-452c-a081-478163c52950 |
+-----+
[root@demo ~(admin)]# openstack port list -c "Fixed IP Addresses"
+-----+
| Fixed IP Addresses
+-----+
| ip_address='172.25.250.254', subnet_id='e9b5e6f0-d484-4598-9ecb-49eb19f6904f' |
```

```

+-----+
[root@demo ~(admin)]# openstack port show \
ec95a76e-3471-452c-a081-478163c52950
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| allowed_address_pairs | |
| binding_host_id | None |
| binding_profile | None |
| binding_vif_details | None |
| binding_vif_type | None |
| binding_vnic_type | normal |
| created_at | 2018-06-22T07:53:48Z |
| data_plane_status | None |
| description | |
| device_id | f0785aec-f535-46f2-b93b-0acff4e951ef |
| device_owner | network:router_interface |
| dns_assignment | None |
| dns_name | None |
| extra_dhcp_opts | |
| fixed_ips | ip_address='172.25.250.254', subnet_id='e9b5...904f' |
| id | ec95a76e-3471-452c-a081-478163c52950 |
| ip_address | None |
| mac_address | fa:16:3e:44:1e:9d |
| name | |
| network_id | 95a56df3-a710-4bc6-b7c5-fc5d3e725df6 |
| option_name | None |
| option_value | None |
| port_security_enabled | False |
| project_id | ddc7e84587e84e90afe1c1f6b2ccd86d |
| qos_policy_id | None |
| revision_number | 10 |
| security_group_ids | |
| status | ACTIVE |
| subnet_id | None |
| tags | |
| trunk_details | None |
| updated_at | 2018-06-22T07:53:52Z |
+-----+-----+

```

- Network ports are created in network namespaces for each interface that the router has. For example, if the router is connected to a private subnet with a range of **10.1.1.0/24**, then the router might have a new interface with an IP address of **10.1.1.1/32** set for a device named **qr-*UUID***. For each network that the router acts as a gateway for, a network device is created in the namespace with a name of **qg-*UUID***.

```

[root@demo ~(admin)]# ip netns exec qrouter-UUID ip a
...output omitted...
13: qr-a669ab8c-1d: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN qlen 1000
    link/ether fa:16:3e:67:22:60 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.1/24 brd 192.168.3.255 scope global qr-a669ab8c-1d
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe67:2260/64 scope link
        valid_lft forever preferred_lft forever

```

```

14: qg-e4929eb0-cd: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN qlen 1000
    link/ether fa:16:3e:e5:15:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.62/24 brd 172.25.250.255 scope global qg-e4929eb0-cd
        valid_lft forever preferred_lft forever
    inet 172.25.250.67/32 brd 172.25.250.67 scope global qg-e4929eb0-cd
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fee5:15b4/64 scope link
        valid_lft forever preferred_lft forever

```

- A set of Netfilter rules are created, which ensure the routing of incoming and outgoing traffic.

```

[root@demo ~(admin)]# ip netns exec qrouter-UUID iptables -L -nv -t nat
Chain PREROUTING (policy ACCEPT 0 packets, 0 bytes)
pkts bytes target     prot opt in     out      source          destination
      0     0 neutron-l3-agent-PREROUTING  all   --  *       *           0.0.0.0/0
      0     0.0.0.0/0

... output omitted ...

Chain neutron-l3-agent-OUTPUT (1 references)
pkts bytes target     prot opt in     out      source          destination
      0     0 DNAT       all   --  *       *           0.0.0.0/0
      0     0.0.0.0/0 to:192.168.3.10

... output omitted ...

Chain neutron-l3-agent-PREROUTING (1 references)
pkts bytes target     prot opt in     out      source          destination
      0     0 DNAT       all   --  *       *           0.0.0.0/0
      0     0.0.0.0/0 to:192.168.3.10

... output omitted ...

```

- A routing table is created in the network namespace, which defines how the traffic should be handled.

```

[root@demo ~(admin)]# ip netns exec qrouter-UUID route
Kernel IP routing table
Destination     Gateway         Genmask        Flags Metric Ref  Use Iface
default         workstation.lab 0.0.0.0        UG    0      0      0
    qg-e4929eb0-cd
172.25.250.0   0.0.0.0        255.255.255.0  U      0      0      0
    qg-e4929eb0-cd
192.168.3.0    0.0.0.0        255.255.255.0  U      0      0      0
    qr-a669ab8c-1d

```

Managing Routers Using the Dashboard

The following steps outline the process for managing routers in OpenStack using the Dashboard.

- Log in to the Dashboard. Within the Project tab, navigate to Network → Routers.
- Click + Create Router to create a new router. Give the router a name and click Create Router.

3. Click the name of the router to access its details. Click the **Interfaces** tab to manage the interfaces for the router.
4. Click **+ Add Interface** to add a new interface. Connect the router to a subnet by selecting it from the Subnet list and click **Submit**.
5. Click **Delete Interface** for each entry to disconnect the router from the subnet.
6. Click **Set Gateway** to define a gateway for the router. From the External Network list, select an external network and click **Submit**.
7. From the Routers tab, the gateway can be removed by clicking **Clear Gateway**. From the drop-down associated with the router, click **Delete Router** to delete the router.

Managing Routers Using the Command-Line Interface

The following steps outline the process for managing routers in OpenStack using the command-line interface.

1. To create a router, source the identity environment file for the user and run the **openstack router create** command.
2. To connect the router to a private network, run the **openstack router add subnet** command.
3. To define the router as a gateway for an external network, run the **neutron router-gateway-set** command.
4. To delete an interface from a router, run the **neutron router-interface-delete**.
5. To remove the router as a gateway for a subnet, run the **neutron router-gateway-clear** command.
6. To delete a router, run the **openstack router delete** command.

Reviewing the Implementation of Routers

The following steps outline the process for reviewing the router implementation.

1. From the server, list the network namespaces using the **ip** command. Notice the UUID which corresponds to that of the router.
2. List the network devices present in the namespace. Notice the names of the interface: **qr** indicates that the port is an interface, and **qg** indicates that the port is a gateway.
3. Use the **ping** command to reach the DNS name server through the network port defined as a gateway.



REFERENCES

Further information is available in the *Networking Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

Further information is available in the *Network Functions Virtualization Product Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/network_functions_virtualization_product_guide/

PREPARING OPENSTACK ROUTERS

In this exercise, you will create and configure a router and use command-line tools to test the connectivity of the router.

OUTCOMES

You should be able to:

- Create a router using the dashboard.
- Manage routers and networks using the **openstack** command.
- Check the connectivity of the router using command-line tools.

Log in to **workstation** as **student** using **student** as the password. Run the **lab prep-deploy-router setup** command, which ensures that the required users and projects are present and that the identity environment files are present. The script also ensures that the network environment is set for the exercise.

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

- ▶ 1. Create a router named **finance-router1** using the dashboard as the **developer1** user. Add an interface to the router using the existing **finance-network1** network.
 - 1.1. From **workstation**, open a web browser and navigate to `http://dashboard.overcloud.example.com`. Log in using **developer1** as the username, and **redhat** as the password.
 - 1.2. Within the Project tab, navigate to Network → Routers.
 - 1.3. Click + Create Router to create a new router. Enter **finance-router1** as the Router Name. From the External Network list, select provider-datacentre and click Create Router.
 - 1.4. Click the name of the router, **finance-router1** to access its details. Click the Interfaces tab to manage the interfaces for the router.
 - 1.5. Click + Add Interface to add a new interface. From the Subnet list, select **finance-subnet1** as the subnet and click Submit.
- ▶ 2. From **workstation**, use the **developer1** user to delete the **finance-router1** router and the private network **finance-network1**.
 - 2.1. Open a terminal on **workstation** and source the identity environment for the **developer1** user, located at `/home/student/developer1-finance-rc`.

```
[student@workstation ~]$ source ~/developer1-finance-rc  
[student@workstation ~(developer1-finance)]$
```

- 2.2. Delete the interface in the subnet **finance-subnet1** for the router **finance-router1**.

```
[student@workstation ~ (developer1-finance)]$ openstack router remove subnet \
finance-router1 finance-subnet1
```

2.3. Unset the **provider-datacentre** network as the gateway for the router.

```
[student@workstation ~ (developer1-finance)]$ openstack router unset \
--external-gateway finance-router1
```

2.4. Delete the **finance-router1** router.

```
[student@workstation ~ (developer1-finance)]$ openstack router delete \
finance-router1
```

2.5. Delete the **finance-network1** network.

```
[student@workstation ~ (developer1-finance)]$ openstack network delete \
finance-network1
```

- 3. As the **developer1** user, create the router **finance-router2**. Set the **provider-datacentre** network as the gateway for the router and add an interface in the existing **finance-subnet2** subnet.

3.1. Create a router named **finance-router2**.

```
[student@workstation ~ (developer1-finance)]$ openstack router create \
finance-router2
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-22T10:57:44Z |
| description | |
| distributed | False |
| external_gateway_info | None |
| flavor_id | None |
| ha | False |
| id | 2d7f0378-c0d4-425a-bfe3-edda8e51e12b |
| name | finance-router2 |
| project_id | e6b7f37fadf943e1aae06a5c2e55d646 |
| revision_number | 1 |
| routes | |
| status | ACTIVE |
| tags | |
| updated_at | 2018-06-22T10:57:44Z |
+-----+-----+
```

3.2. Connect the router to the **finance-subnet2** subnet.

```
[student@workstation ~ (developer1-finance)]$ openstack router add subnet \

```

finance-router2 finance-subnet2

- 3.3. Set the external network **provider-datacentre** as the gateway for the router.

```
[student@workstation ~](developer1-finance)]$ openstack router set \
--external-gateway provider-datacentre \
finance-router2
```

- 4. Review the implementation of the router in the **controller0** virtual machine.

- 4.1. Use **ssh** to connect to the **controller0** virtual machine. Use the **sudo -i** command to become root.

```
[student@workstation ~](developer1-finance)]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

- 4.2. List the network namespaces using the **ip netns** command. Notice that the UUID corresponds to the router created in an earlier step.

```
[root@controller0 ~]# ip netns list
qrouter-2d7f0378-c0d4-425a-bfe3-edda8e51e12b
...output omitted...
```

- 4.3. List the network devices present in the namespace. The output indicates that the router is connected to the private subnet using the IP address **192.168.2.1**, and to the public subnet using the IP address **172.25.250.N**. Notice the port named **qr**, which indicates a router interface, and the port named **qg** which indicates a gateway.

```
[root@controller0 ~]# ip netns exec \
qrouter-d09d39ec-d787-463f-a0eb-a8b50974e6ea ip a
42: qr-952e3ebf-66: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:1b:02:b3 brd ff:ff:ff:ff:ff:ff
    inet 192.168.2.1/24 brd 192.168.2.255 scope global qr-952e3ebf-66
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe1b:2b3/64 scope link
        valid_lft forever preferred_lft forever
44: qg-84cc8b4a-bc: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN group default qlen 1000
    link/ether fa:16:3e:a2:b8:68 brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.N/24 brd 172.25.250.255 scope global qg-84cc8b4a-bc
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fea2:b868/64 scope link
        valid_lft forever preferred_lft forever
```

- 4.4. Use the **ping** command to reach the DNS name server, which has the IP address of **172.25.250.254** through the network port defined as a gateway.

```
[root@controller0 ~]# ip netns exec \
qrouter-2d7f0378-c0d4-425a-bfe3-edda8e51e12b \
ping -c3 -I qg-84cc8b4a-bc 172.25.250.254
PING 172.25.250.254 (172.25.250.254) from 172.25.250.N qg-4654fa8b-42: 56(84)
bytes of data.
64 bytes from 172.25.250.254: icmp_seq=1 ttl=64 time=6.95 ms
```

```
64 bytes from 172.25.250.254: icmp_seq=2 ttl=64 time=0.868 ms
64 bytes from 172.25.250.254: icmp_seq=3 ttl=64 time=0.856 ms

--- 172.25.250.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.856/2.893/6.957/2.873 ms
```

4.5. Exit from the **controller0** virtual machine.

```
[root@controller0 ~]# exit
[heat-admin@controller0 ~]$ exit
[student@workstation ~(developer1-finance)]$
```

▶ 5. Log out from dashboard.

Cleanup

From **workstation**, run the **lab prep-deploy-router** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab prep-deploy-router cleanup
```

This concludes the guided exercise.

MAINTAINING FLOATING IP ADDRESSES

OBJECTIVES

After completing this section, students should be able to:

- Discuss what floating IPs are and how they are used in OpenStack
- Manage floating IPs in OpenStack

FLOATING IP ADDRESSES

In OpenStack terminology, a floating IP is an IP address allocated from a pool for a network marked as *external*. A floating IP is a routable IP address which is publicly reachable. Floating IPs enable communication from the external network to the instance. Administrators can associate a floating IP to an instance after it is launched. After a floating IP is associated to an instance, administrators can manage it on the fly. They can, for example, disassociate the floating IP and associate a new one. The OpenStack Networking Service automatically updates the related entries, such as the routing rules, the ports, as well as the Netfilter rules.

By default, the project limit for floating IPs is 50. When a floating IP is disassociated, it becomes available in the pool of floating IP allocated with the external network, and can be attached to another instance.



NOTE

In order to deallocate a floating IP address from a project, the IP address must be disassociated from an instance and released from the pool. When a floating IP is released, there is no guarantee that the same IP will be allocated to the project again.

Floating IPs provide the following benefits:

- Exposing a service running inside an instance, such as for a web server.
- Managing security groups to create advanced network access management. Administrators can allocate floating IPs to a pool of database servers and create rules to restrict access to the replication network.
- Floating IPs can be dynamically associated and disassociated with instances; administrators can both provide and remove access to a public service in seconds.
- High availability solution by programmatically interacting with the OpenStack networking service API to associate and disassociate floating IPs.

L3 ROUTING AND NAT

Network Address Translation (NAT) capabilities are provided by the Neutron L3 agent, which manages the routers as well as the floating IPs. The service generates a set of routing rules to create a static one-to-one mapping from a floating IP on the external network to the private IP that an instance uses. The Neutron L3 agent interacts with the Netfilter service in order to create a routing topology for the floating IPs. Floating IPs are not directly associated with instances; rather, a floating IP is an IP address attached to an OpenStack networking virtual device. Netfilter rules to route traffic from this device to another virtual device attached to an instance.

The following shows some of the Netfilter rules created for a network namespace which handles the network routing from the floating IP to the private IP of an instance. In this example, **172.25.250.67** is the floating IP associated to an instance with a private IP address of **192.168.3.10**.

```
[root@demo ~]# ip netns exec qrouter-cd3fc9d1-25f8-4bd5-b781-1f773331db0c ip addr show
13: qr-a669ab8c-1d: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN qlen 1000
    link/ether fa:16:3e:67:22:60 brd ff:ff:ff:ff:ff:ff
    inet 192.168.3.1/24 brd 192.168.3.255 scope global qr-a669ab8c-1d
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fe67:2260/64 scope link
        valid_lft forever preferred_lft forever
14: qg-e4929eb0-cd: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1450 qdisc noqueue state UNKNOWN qlen 1000
    link/ether fa:16:3e:e5:15:b4 brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.62/24 brd 172.25.250.255 scope global qg-e4929eb0-cd
        valid_lft forever preferred_lft forever
    inet 172.25.250.67/3 brd 172.25.250.67 scope global qg-e4929eb0-cd
        valid_lft forever preferred_lft forever
    inet6 fe80::f816:3eff:fee5:15b4/64 scope link
        valid_lft forever preferred_lft forever
[root@demo ~]# ip netns exec qrouter-cd3fc9d1-25f8-4bd5-b781-1f773331db0c iptables -L -nv -t nat | grep 172.25.250.67
0      0 DNAT      all  --  *      *      0.0.0.0/0          172.25.250.67
      to:192.168.3.10
0      0 DNAT      all  --  *      *      0.0.0.0/0          172.25.250.67
      to:192.168.3.10
0      0 SNAT      all  --  *      *      192.168.3.10       0.0.0.0/0
      to:172.25.250.67
[root@demo ~]# ip netns exec qrouter-cd3fc9d1-25f8-4bd5-b781-1f773331db0c iptables -L -nv -t nat | grep 192.168.3.10
0      0 DNAT      all  --  *      *      0.0.0.0/0          172.25.250.67
      to:192.168.3.10
0      0 DNAT      all  --  *      *      0.0.0.0/0          172.25.250.67
      to:192.168.3.10
0      0 SNAT      all  --  *      *      192.168.3.10       0.0.0.0/0
      to:172.25.250.67
```

Managing Floating IPs in OpenStack

In OpenStack, administrative users can allocate a specific floating IP, whereas non-administrative users can only query floating IPs, which are randomly assigned. In both cases, administrators can either use the **openstack** command-line interface or the **neutron** command-line interface, with the name of the external network as an argument. OpenStack reads the *allocation pool* specified during the creation of the subnet to determine the floating IP to allocate. If administrators request an explicit floating IP already allocated or outside the allocation pool, OpenStack networking triggers an error. The OpenStack Networking Service uses an internal database to maintain the state of the available and allocated floating IPs.

Managing Floating IPs Using the Dashboard

The following steps outline the process for managing floating IPs using the Dashboard.

1. Log in to the Dashboard and navigate to Network → Floating IPs . To create a new floating IP address click on Allocate IP To Project. The external network is automatically selected as the Pool. Click on Allocate IP to create the floating IP.
2. To manage the association of a floating IP to an instance, click Associate for the row of the floating IP address. When the Manage Floating IP Associations window opens, select the network port attached to the instance from the Port to be associated list. Click Associate.
3. To disassociate a floating IP from an instance, navigate to Compute → Instances. Click the arrow next to the Create Snapshot button for the row of the instance and select Disassociate Floating IP. When the Confirm Disassociate Floating IP window appears, click Disassociate Floating IP.
4. To release a floating IP from a project, click the Floating IPs tab in Network menu. Click the arrow next to the Associate button for the row of the floating IP and select Release Floating IP to return the floating IP to the pool. When the Confirm Release Floating IP window appears, click Release Floating IP.

Managing Floating IPs Using the Command-line Interface

The following steps outline the process for managing floating IPs using the command-line interface.

1. To create a specific floating IP from an external network, source the administrative credentials file and run the **openstack floating ip create --floating-ip-address IP external-network** command.
2. To create a floating IP without specifying the address, source the credentials file of a non-administrative user and run the **openstack floating ip create external-network** command.
3. To associate a floating IP to an instance, run the **openstack server add floating ip instance IP** command.

Reviewing the Implementation of Floating IPs

The following steps outline the process for reviewing the implementation of floating IPs in the system.

1. Run the **ip netns list** command to list the network namespaces and locate the router namespace, which starts with **qrouter-** followed by the UUID of the router.
2. Run the **ip netns exec qrouter-UUID ip a** command to list all the IP addresses present in the namespace. Locate the interface named **qg-UUID** to which the floating IP is assigned.
3. Run the **ip netns exec qrouter-UUID iptables -L -nv -t nat | grep floating_IP** command to list all Netfilter rules which manage the routing from the floating IP to the private IP assigned to an instance.



REFERENCES

Further information is available in the *Networking Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

MAINTAINING FLOATING IP ADDRESSES

In this exercise, you will create a set of floating IPs and allocate them to an instance.

OUTCOMES

You should be able to:

- Manage floating IPs using the dashboard.
- Manage floating IPs using the **openstack** unified CLI.
- Check the external connectivity of an instance.

Log in to **workstation** as **student** using **student** as the password. Run the **lab prep-deploy-ips setup** command, which ensures that the required users and projects are present. The script also ensures that the network environment is set for the exercise and that the instance **finance-server1** is running.

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

- ▶ 1. In the dashboard, create a floating IP as the **developer1** user. Associate the floating IP address to the **finance-server1** instance.
 - 1.1. From **workstation**, open a web browser and navigate to `http://dashboard.overcloud.example.com`. Log in using **developer1** as the username, and **redhat** as the password.
 - 1.2. Within the Project tab, navigate to Network → Floating IPs . Click the Allocate IP To Project button. Ensure provider-datacentre is set as the **Pool**. Click Allocate IP.
 - 1.3. Click Associate in the row of the floating IP address. When the Manage Floating IP Associations window opens, select the entry **finance-server1: 192.168.1.5** from the Port to be associated list. Click Associate.
- ▶ 2. Disassociate the floating IP from the instance **finance-server1** to release the address.
 - 2.1. Navigate to Compute → Instances. Click the arrow next to the Create Snapshot button for the row labeled **finance-server1**. Select Disassociate Floating IP to detach the floating IP from the instance. When the Confirm Disassociate Floating IP window appears, click Disassociate Floating IP.
 - 2.2. Navigate to Network → Floating IPs. Click the arrow next to the Associate button for the floating IP row and select Release Floating IP to return the floating IP to the pool. When the Confirm Release Floating IP window appears, click Release Floating IP.
 - 2.3. Log out from dashboard.
- ▶ 3. From **workstation**, open a terminal and source the identity environment of the **architect1** user. Create the floating IP **172.25.250.103** and allocate it to the **finance** project.
 - 3.1. Open a terminal and source the identity environment for the **architect1** user, located at `/home/student/architect1-finance-rc`.

```
[student@workstation ~]$ source ~/architect1-finance-rc
[student@workstation ~(architect1-finance)]$
```

3.2. Create the floating IP **172.25.250.103** in the **provider-datacentre** network.

```
[student@workstation ~(architect1-finance)]$ openstack floating ip create \
--floating-ip-address 172.25.250.103 \
provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| created_at | 2018-06-26T07:09:20Z |
| description | |
| fixed_ip_address | None |
| floating_ip_address | 172.25.250.103 |
| floating_network_id | 95a56df3-a710-4bc6-b7c5-fc5d3e725df6 |
| id | d0147a80-6f42-4011-b191-c3fab63c0f50 |
| name | 172.25.250.103 |
| port_id | None |
| project_id | e6b7f37fadf943e1aae06a5c2e55d646 |
| qos_policy_id | None |
| revision_number | 0 |
| router_id | None |
| status | DOWN |
| subnet_id | None |
| updated_at | 2018-06-26T07:09:20Z |
+-----+-----+
```

► 4. As the **developer1** user, create three floating IP addresses.

4.1. Source the credentials of the **developer1** user.

```
[student@workstation ~(architect1-finance)]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$
```

4.2. Using the command **openstack floating ip create** create one floating IP in the **provider-datacentre** external network.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip \
create provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| created_at | 2018-06-26T07:13:17Z |
| description | |
| fixed_ip_address | None |
| floating_ip_address | 172.25.250.P |
| floating_network_id | 95a56df3-a710-4bc6-b7c5-fc5d3e725df6 |
| id | 2d69c0d6-6b4c-4b01-abb5-e612e23792b6 |
| name | 172.25.250.P |
| port_id | None |
| project_id | e6b7f37fadf943e1aae06a5c2e55d646 |
```

qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-26T07:13:17Z
+-----+-----+	

4.3. Allocate a second floating IP.

[student@workstation ~](developer1-finance)]\$ openstack floating ip \	
create provider-datacentre	
+-----+-----+	
Field	Value
+-----+-----+	
created_at	2018-06-26T07:14:13Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.R
floating_network_id	95a56df3-a710-4bc6-b7c5-fc5d3e725df6
id	9bc614ea-479c-4f5c-9e6e-393f61fb5136
name	172.25.250.R
port_id	None
project_id	e6b7f37fadf943e1aae06a5c2e55d646
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-26T07:14:13Z
+-----+-----+	

4.4. Allocate a third floating IP.

[student@workstation ~](developer1-finance)]\$ openstack floating ip \	
create provider-datacentre	
+-----+-----+	
Field	Value
+-----+-----+	
created_at	2018-06-26T07:14:56Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.N
floating_network_id	95a56df3-a710-4bc6-b7c5-fc5d3e725df6
id	4314781e-088b-494d-b5a8-adb7f0b3d290
name	172.25.250.N
port_id	None
project_id	e6b7f37fadf943e1aae06a5c2e55d646
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-26T07:14:56Z
+-----+-----+	

- 5. As the **developer1** user, associate the third floating IP to the instance **finance-server1**.

5.1. Allocate the third floating IP to the instance **finance-server1**.

```
[student@workstation ~ (developer1-finance)]$ openstack server add \
floating ip finance-server1 172.25.250.N
```

5.2. Ensure that the IP address has been successfully assigned.

```
[student@workstation ~ (developer1-finance)]$ openstack server list \
-c Name -c Networks
+-----+-----+
| Name | Networks |
+-----+-----+
| finance-server1 | finance-network1=192.168.1.S, 172.25.250.N |
+-----+
```

- 6. From **workstation**, use the **ping** and **ssh** commands to test the external connectivity to the instance.

6.1. Use the **ping** command to reach the instance **finance-server1**.

```
[student@workstation ~ (developer1-finance)]$ ping -c 1 172.25.250.N
PING 172.25.250.N (172.25.250.N) 56(84) bytes of data.
64 bytes from 172.25.250.N: icmp_seq=1 ttl=63 time=22.3 ms

--- 172.25.250.N ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 22.369/22.369/22.369/0.000 ms
```

6.2. Initiate a remote connection using the **ssh** command. The connection itself should proceed while the login should be denied, as the private key is missing. This will be discussed further in the next section.

```
[student@workstation ~ (developer1-finance)]$ ssh 172.25.250.N
Warning: Permanently added '172.25.250.N' (ECDSA) to the list of known hosts.
Permission denied (publickey,gssapi-keyex,gssapi-with-mic).
```

- 7. Disassociate the floating IP from the instance **finance-server1** and release the floating IP.

7.1. Disassociate the **172.25.250.N** floating IP from the **finance-server1** instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server remove \
floating ip finance-server1 172.25.250.N
```

7.2. Release the **172.25.250.N** floating IP.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip \
delete 172.25.250.N
```

7.3. List the available floating IPs to confirm the deletion of **172.25.250.N**.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip \
```

```
list -c 'Floating IP Address'  
+-----+  
| Floating IP Address |  
+-----+  
| 172.25.250.103 |  
| 172.25.250.P |  
| 172.25.250.R |  
+-----+
```

Cleanup

From **workstation**, run the **lab prep-deploy-ips** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab prep-deploy-ips cleanup
```

This concludes the guided exercise.

IMPLEMENTING SECURITY

OBJECTIVES

After completing this section, students should be able to:

- Discuss key pairs and security groups
- Manage OpenStack key pairs and security groups

INTRODUCTION TO KEY PAIRS

SSH key pairs allow passwordless secure and trusted access to remote servers. SSH keys are created in a key pair set, comprised of a *private* key and a *public* key. SSH keys are commonly named, for example, `~/.ssh/id_KEY_TYPE.pub` and `~/.ssh/id_KEY_TYPE` where *KEY_TYPE* is an encryption algorithm, such as RSA or DSA. In OpenStack, the filenames will use the name provided when the key pair is created.

The keys in an SSH key pair are related and used in tandem. A resource encrypted or access-protected by one key can only be decrypted or accessed using the companion key. Since one key is public and available to anyone and the other is private and available only to the key owner, the choice of which key to use for encryption determines the behavior.

If the public key is used to protect a resource, anyone can protect that resource, but only the holder of the private key can access the resource. If the private key is used to protect a resource, then only the key owner could have originated the resource, while anyone with the public key can access the resource with confidence that it originated with the key owner. In OpenStack use, instances are deployed with and protected by a public key. Only the private key owner can access such deployed instances.

In OpenStack, SSH private keys are never created or protected by passphrases. Therefore, it is critical for the private key to be protected with proper Linux ownership and permissions.

The private key is used by SSH to perform authentication against the public key on an instance. When successful, access is granted remotely to the deployed instance. When the remote server receives the SSH connection request, it uses the public key to construct and send a challenge. This challenge is an encrypted message that must be decrypted by the private key and a response returned.

The encryption used in SSH key generation cryptography is nearly impossible to brute force decrypt within the expiration time window of the challenge. Unauthorized access to stored private keys, and insufficiently sized keysets can be a risk to security.

If a private key is lost, it cannot be recreated or regenerated from a public key or stored keypair. Instances deployed using the companion public key can no longer be accessed through SSH. The only recourse is to create a new key pair, re-deploy instance with the new public key, and gain SSH access with the new private key.

However, if a key owner still has a private key, but the public key has been lost, the public key can be regenerated from the private key, and then imported to create a stored keypair public key.

IMPLEMENTATION OF KEY PAIRS IN OPENSTACK

OpenStack provides the ability to generate and import existing key pairs. After key generation, the public key is stored in the Compute service database, while the private key is stored locally as needed by the key pair owner. Private keys are never stored in the Compute database.

The screenshot shows a 'Create Key Pair' dialog box. It has a 'Key Pair Name *' input field, a 'Description:' section with explanatory text, and a 'Create Key Pair' button. The URL at the bottom of the browser window is: 15.11.6.13:8430/v3/638630368324008a

Figure 9.1: Creating a SSH key pair

After creating a key pair, use it to connect to an instance. When users import a public key in OpenStack, it is stored in the Compute service database. Importing a public key assumes that the user possesses the private key related to that public key.

The screenshot shows an 'Import Key Pair' dialog box. It has a 'Key Pair Name *' input field, a 'Public Key *' input area, and a 'Description:' section with explanatory text. It also includes command-line examples and a 'Import Key Pair' button. The URL at the bottom of the browser window is: 15.11.6.13:8430/v3/638630368324008a

Figure 9.2: Importing a public key

Managing Key Pairs

Available actions for managing key pairs in OpenStack using the **openstack** command-line interface:

keypair create

Creates a new key pair, both a private and the matching public. The private key must be captured into a file, because a private key cannot be recreated or reproduced from the command line. The public key is stored in the Compute service using the key name specified.

keypair delete

Deletes a key pair (public key) from the Compute service, but does not delete key files.

keypair list

Lists the key pairs currently stored and available to this user in the Compute service.

keypair show

Displays information about a stored key pair (public key) stored in the Compute service.

When a key pair is created from the command line, the public key is stored in the Compute service, as previously stated. The standard output is the private key to be used to access instances deployed with the public key. To capture the private key for repeated use, either redirect the output into a file, or use the **--private-key** option to specify the output filename. The following commands are identical in result:

```
[user@demo ~(user)]$ openstack keypair create .ssh/keypair > private-key.pem  
[user@demo ~(user)]$ openstack keypair create --private-key .ssh/private-key.pem keypair
```

The SSH command is configured to only allow access to remote systems when proper security procedures are being observed. For example, SSH expects the private key to be owned by the user and only readable by that user. To set the correct permissions for a key that a user has created for their own use, that user must set the read and write permission for the file owner, but remove other permissions:

```
[user@demo ~(user)]$ chmod 0600 .ssh/private-key.pem  
[user@demo ~(user)]$ ls -l .ssh/private-key.pem  
-rw----- 1 user user 1679 Jul 2 15:27 private-key.pem
```

When both the matching public and private keys of a key pair exist as files, the public can be imported to the Compute service using the **--public-key** option. The private key (file) will be needed to access instance deployed with that stored key pair.

```
[user@demo ~(user)]$ openstack keypair create --public-key .ssh/public-key.pub keypair
```

Although a private key cannot be recreated, the public key can be recreated from an existing private key file. OpenStack has no command for this, but it is a common command installed with SSH. This command uses the private key file as input and sends the public to standard output. Capture the public key to a file using redirection.

```
[user@demo ~(user)]$ ssh-keygen -y -f ~/.ssh/private-key.pem > ~/.ssh/public-key.pub
```

Accessing Instances

To use SSH to connect to instances using key pairs, images used for instances must be configured to support SSH during the templating process:

- Install the SSH server package, usually provided by the *openssh-server* package. Configure the server to support key-based authentication.
- Prepare **cloud-init** so that the instance can be dynamically configured at deploy time by the Compute service to inject a public key into the instance. The **cloud-init** utility performs the initial instance configuration, such as hostname customization, user creation, SSH public key injection.

When launching an instance, the key pair is specified with the **--key-name KEY_NAME** option.



WARNING

If the private key is lost and instances are deployed with that key pair, SSH access cannot be granted. The instance can be accessed through the dashboard instance console page, or by the VNC console URL, but the only authentication method allowed is password-based. If an account and password is not available for this instance, access is not granted. Although some instances may be configured with a rescue mode, the typical recourse is to redeploy the instances with a new key pair.



NOTE

Most available for Red Hat OpenStack Platform are configured by default with only key-based SSH access via the **cloud-user** user. Images in this classroom also have **root** access enabled with the password **redhat**.

Verifying Key Pairs

To test a key pair instance injection, **ssh** to the instance with the correct private key. Use the **-i** option ("identity") to specify the private key file to use. If a user does not specify a key, or a default in SSH, they instance cannot be accessed.

```
[user@demo ~]$(user)]$ ssh -i PRIVATE_KEY cloud-user@FLOATING_IP
```

The following flowchart shows how the instance reads the private key and ensures it matches an existing public key. If keys match, the connection is granted, otherwise, it is denied.

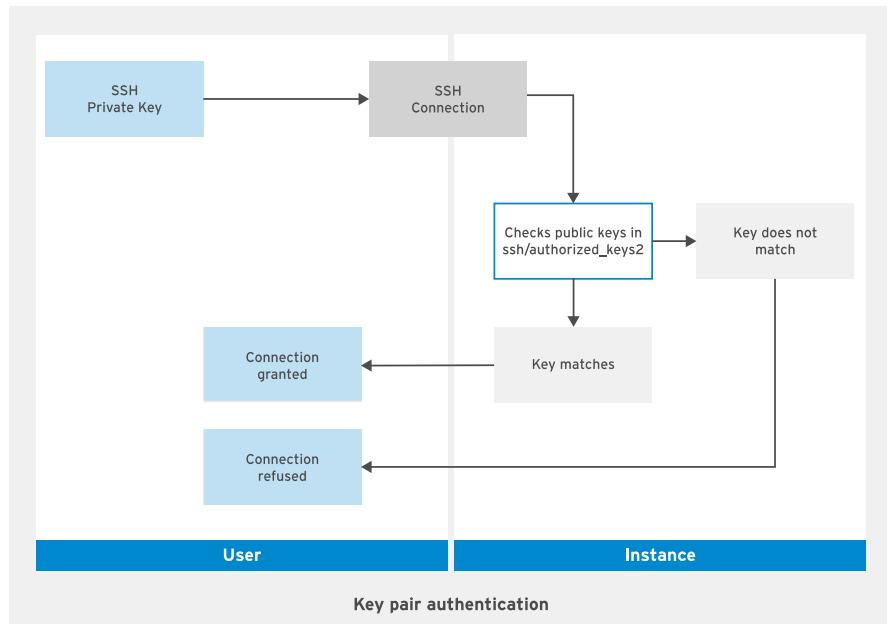


Figure 9.3: Using keys to connect to remote instances

Managing Key Pairs Using the Dashboard

The following steps outline the process for managing key pairs using the Dashboard.

1. Log in the Dashboard and navigate to Project → Compute → Key Pairs.
2. Click +Create Key Pair. When the window Create Key Pair appears, enter a name for the key pair. Click +Create Key Pair to generate the public key and the private key.

3. The next screen displays a message and automatically initiates the download of the private key. A link is also provided to download the private key again. The ability to download that specific private key is lost when the dialogue is closed.

The private key is saved in the **Downloads** directory of the user, because of default browser behavior. Private keys should be moved to the user's **\$HOME/.ssh**.

4. Before being allowed to be used by the SSH command, the private key must be protected. Run the **chmod** command with a mode of **600** to set the proper privileges on the key.
5. To connect to an instance using the private key, use the **ssh** command with the **-i** option. Pass the path of the private key as an argument.
6. To delete a key pair, navigate to Compute → Key Pairs. Click Delete Key Pair in the row for the key pair that needs to be deleted. When the window Confirm Delete Key Pair appears, click Delete Key Pair to confirm the deletion.

Managing Key Pairs Using the Command-Line Interface

The following steps outline the process for managing key pairs using the command-line interface.

1. Source the identity environment of the demo user and run the **openstack keypair create key_pair_name > private_key_path** command. The *private_key_path* indicates a path, such as **/home/student/keys/private-key.pem**.
2. Use the **chmod** command with a mode of **600** to protect the private key.
3. To connect to an instance using the private key, use the **ssh** command with the **-i** option. Pass the path of the private key as an argument.
4. To delete a key pair, run the **openstack keypair delete key_pair_name** command and delete the private key by running the **rm** command.

INTRODUCTION TO SECURITY GROUPS

Security groups are used by OpenStack to provide network packet access control to and from instances. Security groups are packet filtering rules that define the networks and protocols authorized to access the instances. Security groups contain IP filter rules that are applied to instances deployed with that security group. Project members can edit the rules for their default group and add new rules. All projects have a default security group called **default**, which is used at deployment if users do not specify a different security group.

NOTE

By default, the default security group allows all outgoing traffic, and denies all incoming traffic from any source other than VMs in the same security group.

Security groups in the current project are accessed from the dashboard in the Access & Security tab. To access the group rule details, administrators select the **Manage Rules** action for a security group. Creating new security groups is done from the same window using the **Create Security Group** button.

The screenshot shows the Red Hat OpenStack Platform Compute interface. At the top, there are navigation links for Project, Admin, Identity, Compute, Network, Orchestration, Object Store, Overview, Instances, Volumes, Images, and Access & Security. Below this, a breadcrumb trail indicates the current location: Project / Compute / Access & Security / Manage Security Group Rule... The main content area is titled "Manage Security Group Rules: default (ad555f21-58f5-4eb3-9f10-6d7c52ab8018)". It displays a table of security group rules with columns for Direction, Ether Type, IP Protocol, Port Range, Remote IP Prefix, Remote Security Group, and Actions. The table contains six rows of rules, each with a "Delete Rule" button. A message at the bottom left says "Displaying 6 items".

<input type="checkbox"/> Direction	Ether Type	IP Protocol	Port Range	Remote IP Prefix	Remote Security Group	Actions
<input type="checkbox"/> Ingress	IPv4	Any	Any	-	default	<button>Delete Rule</button>
<input type="checkbox"/> Egress	IPv6	Any	Any	::/0	-	<button>Delete Rule</button>
<input type="checkbox"/> Ingress	IPv6	Any	Any	-	default	<button>Delete Rule</button>
<input type="checkbox"/> Egress	IPv4	Any	Any	0.0.0.0/0	-	<button>Delete Rule</button>
<input type="checkbox"/> Ingress	IPv4	ICMP	Any	0.0.0.0/0	-	<button>Delete Rule</button>
<input type="checkbox"/> Ingress	IPv4	TCP	22 (SSH)	0.0.0.0/0	-	<button>Delete Rule</button>

Displaying 6 items

Figure 9.4: Viewing security groups

Security Groups Terminology

TERM	DEFINITION
Security group	The security group is a per-project group that contains rules to manage incoming and outgoing traffic, to and from an OpenStack instances.
Security group rules	Security group rules define how network traffic is filtered. Rules include the protocol to allow or block, and the source networks to allow or deny. Example protocols include SSH, HTTP, HTTPS, and ICMP.
Rule direction	The rule direction specifies whether the rule applies to incoming or outgoing traffic. Valid values are either ingress (incoming) or egress (outgoing).
Remote IP	This option matches the specified IP prefix as the source IP address of the packet to match.
Protocol	This option defines the network protocol to match by the security group rule. Valid values are null , TCP , UDP , and ICMP .
Ports	This option defines the network ports to match by the security group rule. Ports will vary depending on the protocols used.
Ethernet type	This option defines the protocol to match by the security group rule. For TCP traffic, it must be either IPv4 or IPv6. Addresses defined in CIDR notation must match the ingress or egress rules.

Differences Between Firewall and Security Groups

Security groups in OpenStack use the netfilter module by default, which is bundled in all major Linux distributions, to manage security rules. Netfilter analyzes and inspects packets to determine processing. Security groups add security protection similar to a server-based firewall daemon.

Compute nodes are Linux systems that also use netfilter, but administrators will never reconfigure firewall rules on OpenStack nodes. For additional firewall services for peripheral traffic

management, OpenStack offers a **Firewall-as-a-Service** functionality. This component is discussed in a later OpenStack networking course.

MANAGING SECURITY GROUPS

When adding a new security group, operators must set a descriptive name for the security group. This name shows up in the brief description of the instances that use it, whereas the longer description does not. Ideally, the description should indicate which traffic the security group filters. Using clear descriptions make it easier to troubleshoot security group rules. The following lists some of the available actions for managing security groups in OpenStack using the **openstack** command-line interface.

security group create

Create a new security group with default egress rules.

security group delete

Delete a security group and all the rules it contains.

security group list

List the available security group, but without listing all the rules.

security group rule create --protocol protocol

Create a rule, stating the protocol and other optional filtering information.

security group rule list

Lists the security group rules showing the groups to which they belong.

security group show

List detailed security group information.

security group rule show

List detailed security group rule information.

Defining Security Groups

Operators can define new security groups and associated rules with the **openstack** command-line interface or the Dashboard. Create the group first, and then add rules to the group. OpenStack uses both the Compute service and the Networking service to manage security groups and security group rules.

The two services interact with Netfilter, which is a Linux kernel module, to analyze and inspect packets for processing. Netfilter uses the user-defined rules to route the packet through the network stack. The following diagram shows how network packets are handled by Netfilter.

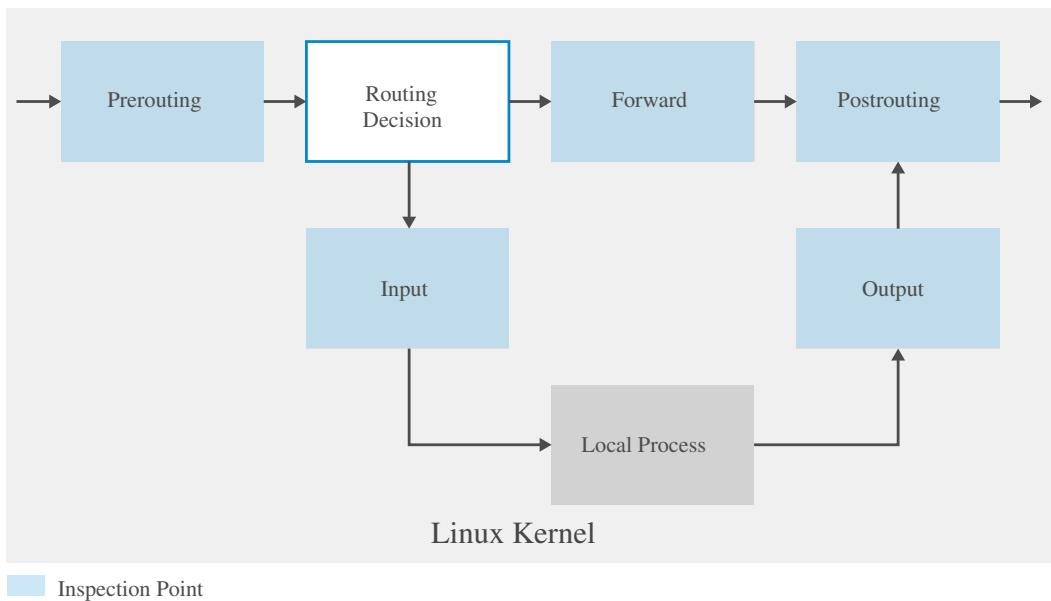


Figure 9.5: Netfilter packet inspection

Describing Security Groups

Operators can use security groups to:

- Define which services consumers can access, and prevent unauthorized access.
- Restrict the remote networks that are authorized to access the instances. Such can be useful for staging environments.
- Further filter access between projects or instances in the same network by specifying a per-group filtering.

Users and applications obey the same security group rules to access other VMs. The following figure illustrates how access is determined by both the original packet source location and the requested destination port. Commonly, virtual machines within a project are intended to work as a cohesive application; security group rules define allowed traffic between instances deployed with the same security group. In the figure, accessing the VMs requires secure HTTPS for public access, but insecure HTTP is allowed between the instances. Using the developers' known IP network address, developers are allowed additional access protocols.

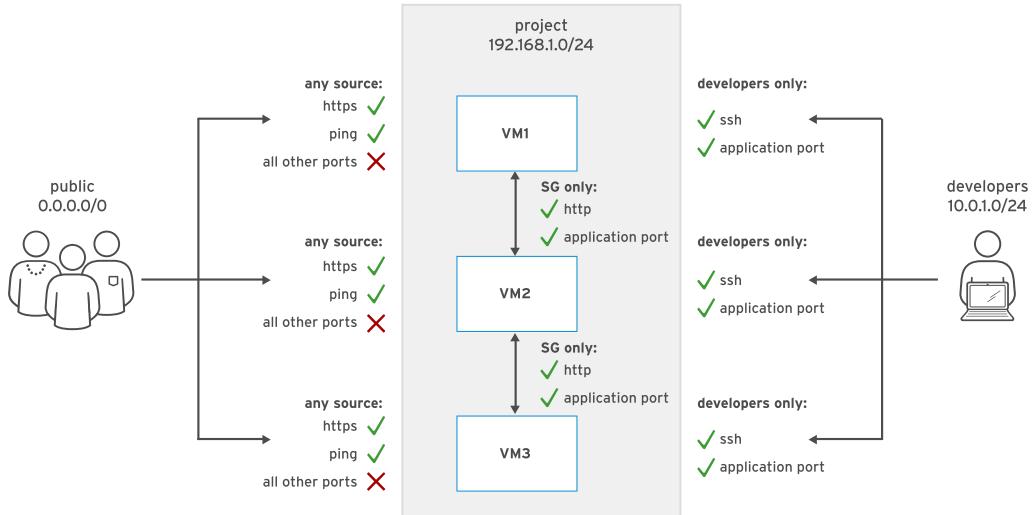


Figure 9.6: Using security rules to define access

This following figure illustrates using security groups to restrict traffic between project VMs to only the protocols necessary for application functions. Public users access the application only by connecting to the web front end. Unauthorized access to the application and database servers is denied. Each security group allows full access by other members in the same security group, but limits other security groups to only the protocols required for VMs to function properly.

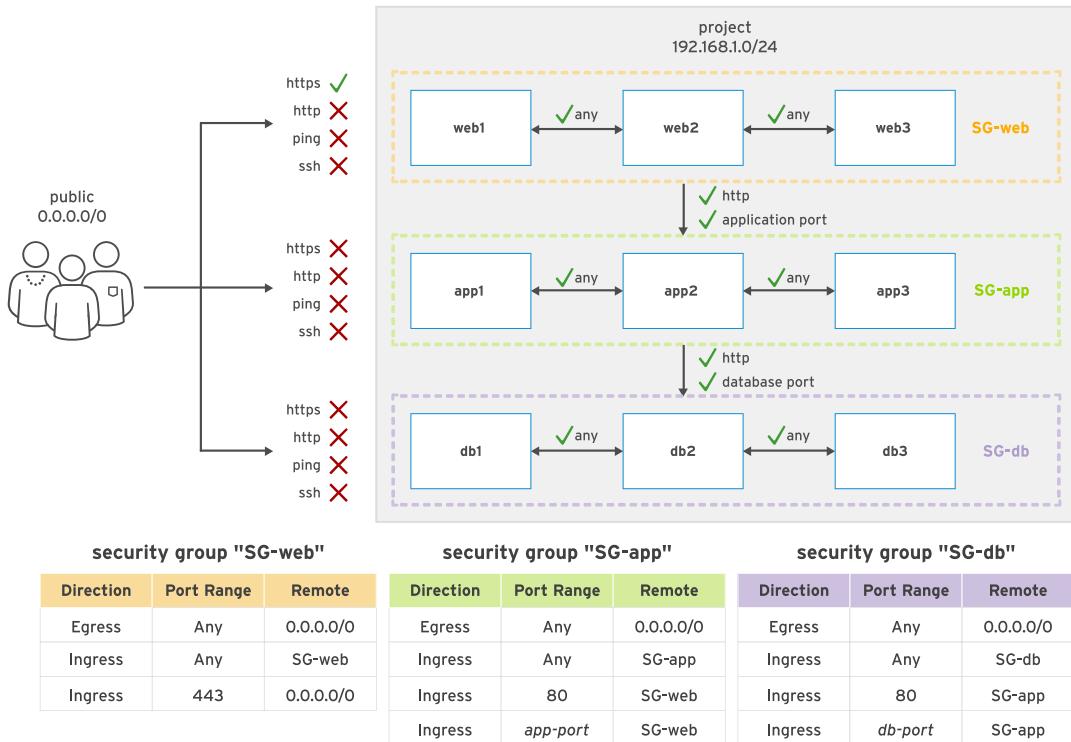


Figure 9.7: Using security rules to manage access

Verifying Security Groups

When an instance is launched and attached to a security group, OpenStack networking sets the security group rules for the system. Security group rules are created with the **iptables** command. The kernel Netfilter module reads and processes every packet to and from the instance. To ensure that the security groups are properly applied, users access a service running on the instance that has a security group rule created to deny access. For example, users can access the web server listening on the port 80 with a rule to deny incoming traffic to port 80.

Managing Security Groups Using the Dashboard

The following steps outline the process for managing security groups using the Dashboard.

- Log in to the Dashboard and navigate to Network → Security Groups.
- To create a new security group, click + Create Security Group. Enter a name for the security group and optionally, a description. Click Create Security Group to confirm the creation of the security group.
- To define rules for the group, click Manage Rules next to the security group. The window lists four security rules:
 - Allows all IPv4 outgoing traffic
 - Allows all IPv6 outgoing traffic.

Click the + Add Rule to define a new rule for the security group. When the window Add Rule appears, select a rule from the Rule list. Select CIDR from the Remote list. Optionally, further restrict the access by specifying a value in the CIDR field. Click Add to create the rule.

- If a rule such as **HTTP** has been added, open a web browser to access the instance by its floating IP. Confirm that the web page displays.

5. To delete a rule, click Delete Rule for the row of the rule. When the window Confirm Delete Rule appears, click Delete Rule.
6. To delete a security group select the check box next to the security group that needs to be deleted. Click on the **down arrow** and choose Delete Security Group. When the window Confirm Delete Security Groups appears, click Delete Security Group to confirm the deletion.

Managing Security Groups Using the Command-Line Interface

The following steps outline the process for managing security groups using the command-line interface.

1. Source the identity environment of a user and run the **openstack security group create security_group** command.
2. To create a new rule in the security group, run the **openstack security group rule create --protocol protocol --dst-port port security_group** command.
--protocol accepts three values: **TCP**, **UDP**, and **ICMP**. If **ICMP** is specified, the **--src-port** and **--dst-port** flags are not supported.
3. If a rule such as **openstack security group rule create --protocol TCP --dst-port 22 security_group** has been added, use the **ssh** command to connect to the instance. Confirm that the connection succeeds.
4. To delete a rule in a security group, run the **openstack security group rule list security_group** command to list all the rules in the group. Copy the UUID of the rule that needs to be removed and run the **openstack security group rule delete UUID** command.
5. To delete a security group, run the **openstack security group delete security_group** command.



REFERENCES

Further information is available in the *Product Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/product_guide/

Further information is available in the *Networking Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

IMPLEMENTING SECURITY

In this exercise, you will manage key pairs and security groups.

OUTCOMES

You should be able to:

- Create a key pair using the dashboard.
- Manage security groups using the dashboard.
- Manage key pairs and security groups using the **openstack** command-line interface.

Log in to **workstation** as **student** using **student** as the password. Run the **lab prep-deploy-sec setup** command, which ensures that the required users and projects are present. The script also ensures that the identity environment files are present.

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

- ▶ 1. As the **developer1** user, create a key pair named **developer1-keypair1** using the dashboard.
 - 1.1. From **workstation**, open a web browser and navigate to `http://dashboard.overcloud.example.com`. Log in using **developer1** as the username, and **redhat** as the password.
 - 1.2. Within the Project tab, navigate to Compute → Key Pairs. Click +Create Key Pair to create a new key pair. When the window Create Key Pair appears, enter **developer1-keypair1** as the Key Pair Name and click +Create Key Pair.
 - 1.3. A window prompting to save the private key file should appear. Select the Save File entry and click OK. The private key will be saved in the **Downloads** directory of the **student** user.
- ▶ 2. As the **developer1** user, create the **finance-secgroup1** security group using the dashboard. Add a rule to allow all ICMP packets to the instance.
 - 2.1. Navigate to Network → Security Groups.
 - 2.2. Click + Create Security Group. When the window Create Security Group appears, enter **finance-secgroup1** as the Name and click Create Security Group.
 - 2.3. Click Manage Rules for the row of the **finance-secgroup1** security group.
 - 2.4. Click + Add Rule to add a new rule in the security group. When the window Add Rule appears, select ALL ICMP from the Rule list and click Add.
 - 2.5. Log out from dashboard.
- ▶ 3. From **workstation**, use the **developer1** credentials to delete the private key associated with the **developer1-keypair1** key pair. Use the **openstack** command to delete the **developer1-keypair1** key pair.

- 3.1. Open a terminal and delete the private key in the **/home/student/Downloads/** directory.

```
[student@workstation ~]$ rm -f ~/Downloads/developer1-keypair1.pem
```

- 3.2. Source the identity environment for the **developer1** user, located at **/home/student/developer1-finance-rc** and delete the key pair **developer1-keypair1**.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack keypair delete \
developer1-keypair1
```

- 4. As the **developer1** user, delete the security group rule that you created as well as the security group.

- 4.1. List the rules in the **finance-secgroup1** security group.

```
[student@workstation ~(developer1-finance)]$ openstack security group rule \
list finance-secgroup1 -c ID -c "IP Protocol"
+-----+-----+
| ID | IP Protocol |
+-----+-----+
| 36f1820e-6f8d-40f7-b768-9072a166c321 | None |
| 619640ff-b4ee-4041-ac6a-ec5725cd82d6 | icmp |
| eb116b13-2ce1-47aa-9cb7-165b8ad18df5 | None |
+-----+-----+
```

- 4.2. Copy the UUID of the **icmp** rule to delete it.

```
[student@workstation ~(developer1-finance)]$ openstack security group rule \
delete 619640ff-b4ee-4041-ac6a-ec5725cd82d6
```

- 4.3. Delete the **finance-secgroup1** security group.

```
[student@workstation ~(developer1-finance)]$ openstack security group \
delete finance-secgroup1
```

- 5. As the **developer1** user, create the key pair **developer1-keypair2** and export the private key to **/home/student/Downloads/developer1-keypair2.pem**. Use the **chmod** command to protect the private key file.

- 5.1. Create the key pair **developer1-keypair2** and save the private key as **/home/student/Downloads/developer1-keypair2.pem**.

```
[student@workstation ~(developer1-finance)]$ openstack keypair create \
developer1-keypair2 > /home/student/Downloads/developer1-keypair2.pem
```

- 5.2. Use the **chmod** command with a mode of **600** to protect the private key.

```
[student@workstation ~(developer1-finance)]$ chmod 600 \
/home/student/Downloads/developer1-keypair2.pem
```

- 6. As the **developer1** user, create the **finance-secgroup2** security group and create rules to allow ICMP and SSH traffic.

6.1. Create the **finance-secgroup2** security group.

```
[student@workstation ~ (developer1-finance)]$ openstack security group \
create finance-secgroup2
+-----+-----+
| Field      | Value           |
+-----+-----+
| created_at | 2018-06-26T09:37:05Z |
| description | finance-secgroup2 |
| id          | a7cab5cc-a12e-453b-bf95-b820d94a8c8a |
| name        | finance-secgroup2 |
| project_id  | e6b7f37fadf943e1aae06a5c2e55d646 |
| revision_number | 2 |
| rules       | created_at='2018-06-26T09:37:05Z',
               direction='egress', ethertype='IPv4',
               id='05b2ce9b-90b1-4962-b51b-58af4da73832',
               updated_at='2018-06-26T09:37:05Z'
               |
               | created_at='2018-06-26T09:37:05Z',
               direction='egress', ethertype='IPv6',
               id='38594c0d-4092-49ed-b312-04a7b556dad8',
               updated_at='2018-06-26T09:37:05Z'
|
| updated_at  | 2018-06-26T09:37:05Z |
+-----+-----+
```

6.2. Add a security group rule in the **finance-secgroup2** security group to allow remote ICMP traffic.

```
[student@workstation ~ (developer1-finance)]$ openstack security group rule \
create \
--protocol icmp \
finance-secgroup2
+-----+-----+
| Field      | Value           |
+-----+-----+
| created_at | 2018-06-26T09:44:35Z |
| description |             |
| direction   | ingress        |
| ether_type  | IPv4            |
| id          | 1933f8dd-1194-4ac8-a0f7-e74a6a63711c |
| name        | None            |
| port_range_max | None          |
| port_range_min | None          |
| project_id  | e6b7f37fadf943e1aae06a5c2e55d646 |
| protocol    | icmp          |
| remote_group_id | None          |
| remote_ip_prefix | 0.0.0.0/0     |
| revision_number | 0              |
| security_group_id | a7cab5cc-a12e-453b-bf95-b820d94a8c8a |
| updated_at   | 2018-06-26T09:44:35Z |
+-----+-----+
```

6.3. Add a security rule to allow remote connections using **SSH**.

```
[student@workstation ~](developer1-finance)]$ openstack security group rule \
create \
--protocol tcp \
--dst-port 22 \
finance-secgroup2
+-----+-----+
| Field      | Value          |
+-----+-----+
| created_at | 2018-06-26T09:45:55Z |
| description |                |
| direction   | ingress         |
| ether_type  | IPv4            |
| id          | e1664aea-cdeb-45bd-85c6-69d7c09b9d2a |
| name        | None            |
| port_range_max | 22 |
| port_range_min | 22 |
| project_id  | e6b7f37fadf943e1aae06a5c2e55d646 |
| protocol    | tcp             |
| remote_group_id | None           |
| remote_ip_prefix | 0.0.0.0/0 |
| revision_number | 0              |
| security_group_id | a7cab5cc-a12e-453b-bf95-b820d94a8c8a |
| updated_at   | 2018-06-26T09:45:55Z |
+-----+-----+
```

Cleanup

From **workstation**, run the **lab prep-deploy-sec** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab prep-deploy-sec cleanup
```

This concludes the guided exercise.

PREPARING TO DEPLOY AN INSTANCE WITH PUBLIC ACCESS

PERFORMANCE CHECKLIST

In this lab, you will create a public network, as well as a router. You will create some floating IPs, a key pair, and various security group rules.

OUTCOMES

You should be able to:

- Create a public network and its associated subnet.
- Create and configure a router.
- Create a key pair.
- Set security group rules in a security group.

Log in to **workstation** as **student** using **student** as the password. Run the **lab prep-deploy-ext-lab setup** command, which verifies that the OpenStack services are running and the resources created in previous sections are available. The lab script removes all floating IP addresses and removes the external network created previously.

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

1. As the **architect1** user, create the external network **provider-datacentre** and subnet **provider-subnet-172.25.250**. Use the following specifications:

OPENSTACK NETWORK RESOURCES	
Resource	Detail
External Network	<ul style="list-style-type: none">• Name: provider-datacentre• Shared: Yes• External: Yes• Provider network type: flat• Provider network name: datacentre• Subnet name: provider-subnet-172.25.250• Network subnet: 172.25.250.0/24• Network gateway: 172.25.250.254• DNS name server: 172.25.250.254• Allocation pool: 172.25.250.101, 172.25.250.189• DHCP: disabled

2. As **operator1**, create the router **production-router1** and connect it to the project subnet, **production-subnet1**.

- As the **architect1** user, set the router **production-router1** as the gateway with an IP address of **172.25.250.125**. Create the floating IP **172.25.250.128** in the **provider-datacentre** network.
- As the **operator1** user, create two additional floating IPs in the **provider-datacentre** network.
- As the **operator1** user, create the security group **production-secgroup1** and add the rules listed in the following table.

OPENSTACK SECURITY RESOURCES

Resource	Detail
Security Group	Security group name: production-secgroup1 Rules: <ul style="list-style-type: none">Protocol: TCP, port 22Protocol: TCP, port 80Protocol: TCP, port 443

- As the **operator1** user, create the key pair **production-keypair1** and save the private key to **/home/student/Downloads/production-keypair1.pem**. Use the **chmod** command to protect the private key file.

Evaluation

From **workstation**, run the **lab prep-deploy-ext-lab grade** command to confirm success of this exercise.

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

Cleanup

From **workstation**, run the **lab prep-deploy-ext-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab prep-deploy-ext-lab cleanup
```

This concludes the lab.

PREPARING TO DEPLOY AN INSTANCE WITH PUBLIC ACCESS

PERFORMANCE CHECKLIST

In this lab, you will create a public network, as well as a router. You will create some floating IPs, a key pair, and various security group rules.

OUTCOMES

You should be able to:

- Create a public network and its associated subnet.
- Create and configure a router.
- Create a key pair.
- Set security group rules in a security group.

Log in to **workstation** as **student** using **student** as the password. Run the **lab prep-deploy-ext-lab setup** command, which verifies that the OpenStack services are running and the resources created in previous sections are available. The lab script removes all floating IP addresses and removes the external network created previously.

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

1. As the **architect1** user, create the external network **provider-datacentre** and subnet **provider-subnet-172.25.250**. Use the following specifications:

OPENSTACK NETWORK RESOURCES	
Resource	Detail
External Network	<ul style="list-style-type: none"> • Name: provider-datacentre • Shared: Yes • External: Yes • Provider network type: flat • Provider network name: datacentre • Subnet name: provider-subnet-172.25.250 • Network subnet: 172.25.250.0/24 • Network gateway: 172.25.250.254 • DNS name server: 172.25.250.254 • Allocation pool: 172.25.250.101,172.25.250.189 • DHCP: disabled

- 1.1. From **workstation**, source the identity environment file for the **architect1** user.

```
[student@workstation ~]$ source ~/architect1-production-rc
[student@workstation ~(architect1-production)]$
```

1.2. Create the external network named **provider-datacentre**.

```
[student@workstation ~-(architect1-production)]$ openstack network create \
--external \
--share \
--provider-network-type flat \
--provider-physical-network datacentre \
provider-datacentre

+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-27T07:19:48Z |
| description | |
| dns_domain | None |
| id | 84f36592-b94b-411a-be5e-811d55335169 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1500 |
| name | provider-datacentre |
| port_security_enabled | True |
| project_id | ddc7e84587e84e90afe1c1f6b2cccd86d |
| provider:network_type | flat |
| provider:physical_network | datacentre |
| provider:segmentation_id | None |
| qos_policy_id | None |
| revision_number | 6 |
| router:external | External |
| segments | None |
| shared | True |
| status | ACTIVE |
| subnets | |
| tags | |
| updated_at | 2018-06-27T07:19:48Z |
+-----+-----+
```

1.3. Create the **provider-subnet-172.25.250** subnet for the external network with an allocation pool of **172.25.250.101-172.25.250.189**. Disable DHCP services for the subnet and use **172.25.250.254** as the gateway as well as the DNS name server.

```
[student@workstation ~-(architect1-production)]$ openstack subnet create \
--subnet-range 172.25.250.0/24 \
--no-dhcp \
--gateway 172.25.250.254 \
--dns-nameserver 172.25.250.254 \
--allocation-pool start=172.25.250.101,end=172.25.250.189 \
--network provider-datacentre \
provider-subnet-172.25.250

+-----+-----+
```

Field	Value
allocation_pools	172.25.250.101-172.25.250.189
cidr	172.25.250.0/24
created_at	2018-06-27T07:27:27Z
description	
dns_nameservers	172.25.250.254
enable_dhcp	False
gateway_ip	172.25.250.254
host_routes	
id	bac706ff-e176-4f97-98f2-1684edcd98fe
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	provider-subnet-172.25.250
network_id	84f36592-b94b-411a-be5e-811d55335169
project_id	ddc7e84587e84e90afe1c1f6b2cccd86d
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-06-27T07:27:27Z

2. As **operator1**, create the router **production-router1** and connect it to the project subnet, **production-subnet1**.

2.1. Source the credentials for the **operator1** user.

```
[student@workstation ~-(architect1-production)]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$
```

2.2. Create the **production-router1** router.

[student@workstation ~(operator1-production)]\$ openstack router create \ production-router1	
Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-27T07:35:48Z
description	
distributed	False
external_gateway_info	None
flavor_id	None
ha	False
id	19792e97-698f-42b5-aa9b-bfd5be667864
name	production-router1
project_id	adcf826c851145638d2f49e9fe56d074
revision_number	1
routes	
status	ACTIVE
tags	

updated_at	2018-06-27T07:35:48Z	
-----+-----+-----+		

2.3. Connect the router to the project subnet, **production-subnet1**.

```
[student@workstation ~-(operator1-production)]$ openstack router add \
subnet production-router1 production-subnet1
```

3. As the **architect1** user, set the router **production-router1** as the gateway with an IP address of **172.25.250.125**. Create the floating IP **172.25.250.128** in the **provider-datacentre** network.

3.1. Source the credentials file for the **architect1** user.

```
[student@workstation ~-(operator1-production)]$ source ~/architect1-production-rc
[student@workstation ~-(architect1-production)]$
```

3.2. Set the router **production-router1** as the gateway for the external network, **provider-datacentre**.

```
[student@workstation ~-(architect1-production)]$ openstack router set \
--external-gateway provider-datacentre \
--fixed-ip ip-address=172.25.250.125 \
production-router1
```

3.3. Create the **172.25.250.128** floating IP in the **provider-datacentre** network.

```
[student@workstation ~-(architect1-production)]$ openstack floating ip create \
--floating-ip-address 172.25.250.128 provider-datacentre
+-----+-----+
| Field          | Value           |
+-----+-----+
| created_at     | 2018-06-27T07:49:00Z |
| description    |                  |
| fixed_ip_address | None            |
| floating_ip_address | 172.25.250.128 |
| floating_network_id | 84f36592-b94b-411a-be5e-811d55335169 |
| id             | f9663984-7e72-4a3d-a094-282141f4edc7 |
| name           | 172.25.250.128 |
| port_id         | None            |
| project_id     | adcf826c851145638d2f49e9fe56d074 |
| qos_policy_id   | None            |
| revision_number | 0               |
| router_id       | None            |
| status          | DOWN            |
| subnet_id       | None            |
| updated_at      | 2018-06-27T07:49:00Z |
+-----+-----+
```

4. As the **operator1** user, create two additional floating IPs in the **provider-datacentre** network.

4.1. Source the credentials of the **operator1** user.

```
[student@workstation ~-(architect1-production)]$ source ~/operator1-production-rc
```

```
[student@workstation ~ (operator1-production)]$
```

4.2. From the floating IP pool **provider-datacentre**, create a floating IP.

```
[student@workstation ~ (operator1-production)]$ openstack floating ip create \
provider-datacentre
```

Field	Value
created_at	2018-06-27T07:52:07Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.N
floating_network_id	84f36592-b94b-411a-be5e-811d55335169
id	4458ee3b-acdc-43c3-b37e-9fb25eaef623
name	172.25.250.N
port_id	None
project_id	adcf826c851145638d2f49e9fe56d074
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-27T07:52:07Z

4.3. Create a second floating IP from the **provider-datacentre** floating IP pool.

```
[student@workstation ~ (operator1-production)]$ openstack floating ip create \
provider-datacentre
```

Field	Value
created_at	2018-06-27T07:53:01Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.P
floating_network_id	84f36592-b94b-411a-be5e-811d55335169
id	f8f66727-05a4-4259-9293-176cd7f3e3a2
name	172.25.250.P
port_id	None
project_id	adcf826c851145638d2f49e9fe56d074
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-27T07:53:01Z

5. As the **operator1** user, create the security group **production-secgroup1** and add the rules listed in the following table.

OPENSTACK SECURITY RESOURCES

Resource	Detail
Security Group	Security group name: production-secgroup1 Rules: <ul style="list-style-type: none">Protocol: TCP, port 22Protocol: TCP, port 80Protocol: TCP, port 443

- 5.1. As the **operator1** user, create the security group **production-secgroup1**

```
[student@workstation ~ (operator1-production)]$ openstack security group create \
production-secgroup1
...output omitted...
```

- 5.2. Add a rule in the security group **production-secgroup1** to allow **ssh** access.

```
[student@workstation ~ (operator1-production)]$ openstack security group \
rule create --protocol tcp --dst-port 22 production-secgroup1
+-----+-----+
| Field      | Value           |
+-----+-----+
| created_at | 2018-06-27T07:57:13Z |
| description |                 |
| direction   | ingress          |
| ether_type  | IPv4             |
| id          | 9f4a3e64-1b0f-4815-a00e-1d84808a55ac |
| name        | None             |
| port_range_max | 22               |
| port_range_min | 22               |
| project_id  | adcf826c851145638d2f49e9fe56d074 |
| protocol    | tcp               |
| remote_group_id | None            |
| remote_ip_prefix | 0.0.0.0/0       |
| revision_number | 0                |
| security_group_id | 90d1cdb7-85fb-4e3f-891b-b8f036e300ce |
| updated_at   | 2018-06-27T07:57:13Z |
+-----+-----+
```

- 5.3. Add a security rule to allow **HTTP** connections using a default port of 80.

```
[student@workstation ~ (operator1-production)]$ openstack security group \
rule create --protocol tcp --dst-port 80 production-secgroup1
+-----+-----+
| Field      | Value           |
+-----+-----+
| created_at | 2018-06-27T07:57:43Z |
| description |                 |
| direction   | ingress          |
| ether_type  | IPv4             |
| id          | 9f4a3e64-1b0f-4815-a00e-1d84808a55ac |
| name        | None             |
| port_range_max | 80               |
| port_range_min | 80               |
| project_id  | adcf826c851145638d2f49e9fe56d074 |
| protocol    | tcp               |
| remote_group_id | None            |
| remote_ip_prefix | 0.0.0.0/0       |
| revision_number | 1                |
| security_group_id | 90d1cdb7-85fb-4e3f-891b-b8f036e300ce |
| updated_at   | 2018-06-27T07:57:43Z |
+-----+-----+
```

ether_type	IPv4
id	1eca2a86-03ec-486d-84cd-2a49f4d1e244
name	None
port_range_max	80
port_range_min	80
project_id	adcf826c851145638d2f49e9fe56d074
protocol	tcp
remote_group_id	None
remote_ip_prefix	0.0.0.0/0
revision_number	0
security_group_id	90d1cdb7-85fb-4e3f-891b-b8f036e300ce
updated_at	2018-06-27T07:57:43Z
+	-----+-----+

5.4. Add a security rule to allow **HTTPS** connections using the default port of 443.

[student@workstation ~-(operator1-production)]\$ openstack security group \rule create --protocol tcp --dst-port 443 production-secgroup1	+	-----+-----+
Field	Value	
+	-----+-----+	-----+-----+
created_at	2018-06-27T08:00:19Z	
description		
direction	ingress	
ether_type	IPv4	
id	40018b0c-1100-414c-8489-adbf69e47083	
name	None	
port_range_max	443	
port_range_min	443	
project_id	adcf826c851145638d2f49e9fe56d074	
protocol	tcp	
remote_group_id	None	
remote_ip_prefix	0.0.0.0/0	
revision_number	0	
security_group_id	90d1cdb7-85fb-4e3f-891b-b8f036e300ce	
updated_at	2018-06-27T08:00:19Z	
+	-----+-----+	-----+-----+

6. As the **operator1** user, create the key pair **production-keypair1** and save the private key to **/home/student/Downloads/production-keypair1.pem**. Use the **chmod** command to protect the private key file.

6.1. Create the key pair **production-keypair1** and save the private key as **/home/student/Downloads/production-keypair1.pem**.

```
[student@workstation ~-(operator1-production)]$ openstack keypair create \
production-keypair1 > /home/student/Downloads/production-keypair1.pem
```

6.2. Use the **chmod** command with a mode of **600** to protect the private key.

```
[student@workstation ~-(operator1-production)]$ chmod 600 \
/home/student/Downloads/production-keypair1.pem
```

Evaluation

From **workstation**, run the **lab prep-deploy-ext-lab grade** command to confirm success of this exercise.

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

Cleanup

From **workstation**, run the **lab prep-deploy-ext-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab prep-deploy-ext-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The OpenStack Networking Service is the networking framework for OpenStack. OpenStack networking provides a set of services which aim to provide an interface for defining network connectivity and network addressing in OpenStack environments.
- The original OpenStack network implementation, **nova-network**, assumed a basic networking model where network isolation was performed through the use of VLANs and Netfilter. OpenStack networking now uses plug-ins to provide a pluggable API back end.
- External networking allows instances to access networks outside their private range, such as the Internet. This type of network uses network address translation (NAT). For external traffic to reach the instance, administrators must create the security group rules to network allow traffic such as SSH or ICMP to reach an instance.
- For instances to communicate with any external subnet, a router must be deployed. The Red Hat OpenStack Platform provides such routing capabilities via Software-defined Networking (SDN). SDN-based virtual routers are similar to physical routers.
- In OpenStack terminology, a floating IP is an IP address allocated from a pool for a network marked as external. A floating IP is a routable IP address that is publicly reachable. Floating IPs enable communication from the external network to an instance. Administrators can associate a floating IP to an instance after it is launched.
- OpenStack provides the ability to generate and import existing key pairs.
- Security groups are used by OpenStack to provide access control to and from the instances.

DEPLOYING AN INSTANCE WITH PUBLIC ACCESS

GOAL

Launch and verify an instance with public access.

OBJECTIVES

- Launch an instance with the additional OpenStack resources required to have public access.
- Verify an instance with public access.
- Manage multi-tenant networking.

SECTIONS

- Launching an Instance with Public Access (and Guided Exercise)
- Verifying an Instance with Public Access (and Guided Exercise)
- Managing Multi-tenant Networking (and Quiz)

LAB

- Deploying an Instance with Public Access

LAUNCHING AN INSTANCE WITH PUBLIC ACCESS

OBJECTIVE

After completing this section, students should be able to launch an instance with the additional OpenStack resources required to have public access.

DEFINING AN INSTANCE WITH PUBLIC ACCESS

An instance with public access is an instance that has access to the external network. The previous chapter discussed the common items needed for an instance with public access, and this chapter will discuss launching and verifying the functionality of such an instance. Launching an instance with public access is very similar to launching a private instance in a project. A self-service user can use the same **openstack server create** command (with the image, flavor, private network, and so on). Launching an instance with public access has security implications, so there are a few extra options to use with the **openstack server create** command.

Key pairs

The option to include a key pair at instance creation time is **--key-name *keypair-name***. The *keypair-name* key pair *must* be created before the instance is launched. It is very important that the private SSH key is kept safe because it is used later to log in to the instance without a password. The following example shows an instance being created using the key pair named **demo**.

```
[user@demo ~]$(user-demo)]$ openstack server create --flavor default \
--image rhel7 --key-name demo demo-server1
```

After the instance is running, the private SSH key can be used to connect to the instance. Normally, the **cloud-user** user is automatically created in the instance, and the **root** user is not allowed to log in through SSH. The following example shows an **ssh** command connecting to an instance using a floating IP address.

```
[user@demo ~]$(user-demo)]$ ssh -i ~/Downloads/demo.pem cloud-user@1.1.1.1
[cloud-user@demo-server1 ~]$
```



NOTE

It is a recommended practice to name the SSH private key using the same name as the OpenStack key pair. This makes it simpler to determine which SSH private key belongs to which OpenStack key pair. Otherwise you must match the fingerprint of the private key with the key pair in OpenStack (using **openstack keypair show *keypair-name***).

To determine which (if any) key pair was used for an instance, use the **openstack server show** command. The **key_name** entry in the output shows the key name used for the instance. The following example shows that **demo-server1** uses the key pair named **demo**.

```
[user@demo ~]$(user-demo)]$ openstack server show demo-server1
+-----+
+-----+
```

Field	Value
+-----+-----+	
OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2018-06-12T15:46:34.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	demo-network1=192.168.1.N, 1.1.1.1
config_drive	
created	2018-06-12T15:46:19Z
flavor	default (db97c88d-36a3-4ecc-b66f-7d99b59f1fbc)
hostId	2e30...aefb8
id	81519aa2-b3af-4a05-9e0d-81b1660a16b4
image	rhel7 (2cb5074b-aa91-4240-9d97-a4b5c0079853)
key_name	demo
name	demo-server1
progress	0
project_id	1bfc1e465c0442aab6d5fa387ae1c244
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-13T08:27:09Z
user_id	c0a85a6918b14200884a276d92415872
volumes_attached	

An SSH key that you create outside Red Hat OpenStack Platform can also be used by OpenStack. When you use the **openstack keypair create** command to create an OpenStack key pair, you can give your existing public key through the **--public-key *public-ssh-key-file*** option. The following example shows the creation of a key pair named **demo** that uses the existing public key file named **id_rsa.pub**.

```
[user@demo ~(user-demo)]$ openstack keypair create \
--public-key ~/ssh/id_rsa.pub demo
+-----+
| Field      | Value
+-----+
| fingerprint | 26:c8:27:45:c1:71:d3:bd:9e:fe:2a:a2:3e:b8:64:5f |
| name       | demo
| user_id    | c0a85a6918b14200884a276d92415872
+-----+
```

Security groups

The option to include a security group at instance creation is **--security-group *security-group-name***. All the rules in the security group apply to the instance. Thus, if a security group includes access to SSH (TCP/22) and HTTP (TCP/80), those ports are opened, regardless whether the instance is running the services on those ports or not. The following example shows an instance being created using the security group named **secgroup1**.

```
[user@demo ~(user-demo)]$ openstack server create --flavor default \
--image rhel7 --security-group secgroup1 demo-server1
```

Unlike a key pair, a security group can be created and attached to an instance *after* the instance has been launched, and multiple security groups can be associated with an instance. To add a security group to a running instance, use the **openstack server add security group** command. The following example shows the security group named **secgroup1** being added to a running instance named **demo-server1**.

```
[user@demo ~(user-demo)]$ openstack server add security group demo-server1
secgroup1
```

To remove a security group from an instance, use the **openstack server remove security group *instance security_group*** command.

To determine which security groups are used by an instance, use the **openstack server show** command. The **security_groups** entry in the output shows the security group names used by the instance. The following example shows that the security groups **secgroup1** and **default** are used by **demo-server1**.

```
[user@demo ~(user-demo)]$ openstack server show demo-server1
+-----+
+-----+
| Field          | Value
|               |
+-----+
```

```

| OS-DCF:diskConfig           | MANUAL
|
| OS-EXT-AZ:availability_zone | nova
|
| OS-EXT-STS:power_state     | Running
|
| OS-EXT-STS:task_state      | None
|
| OS-EXT-STS:vm_state        | active
|
| OS-SRV-USG:launched_at    | 2018-06-12T15:46:34.000000
|
| OS-SRV-USG:terminated_at   | None
|
| accessIPv4                 |
|
| accessIPv6                 |
|
| addresses                  | demo-network1=192.168.1.N, 1.1.1.1
|
| config_drive               |
|
| created                    | 2018-06-12T15:46:19Z
|
| flavor                      | default (db97c88d-36a3-4ecc-b66f-7d99b59f1fbc)
|
| hostId                     | 2e30...aefb8
|
| id                          | 81519aa2-b3af-4a05-9e0d-81b1660a16b4
|
| image                       | rhel7 (2cb5074b-aa91-4240-9d97-a4b5c0079853)
|
| key_name                   | None
|
| name                        | demo-server1
|
| progress                    | 0
|
| project_id                 | 1bfc1e465c0442aab6d5fa387ae1c244
|
| properties                  |
|
| security_groups             | name='secgroup1'
|
|                               | name='default'
|
| status                      | ACTIVE
|
| updated                     | 2018-06-13T08:27:09Z
|
| user_id                     | c0a85a6918b14200884a276d92415872
|
| volumes_attached            |
+
+-----+
+-----+

```

Security group rules can be added to or removed from a security group and the effect is immediate. As soon as a rule for SSH (TCP/22) is added to a security group, all instances using that security group open the TCP port 22. This can be convenient if there is a need to change access to instances immediately, but it could have unintended consequences. When adding or removing a security group rule, any instance that uses that security group will be immediately updated.

It is recommended that security groups remain relatively small to avoid these unintended consequences. For instance, a security group named **SSH** could be created to allow only SSH (TCP/22). Another security group named **HTTPS** could be created to allow only HTTPS (TCP/443). Yet another security group named **ICMP** could be created to allow only ICMP packets to get through to the instance. A user could then assign one or more of these security groups to an instance, and it would be clear which ports are allowed through to the instance.

If you don't specify a security group when you launch an instance, OpenStack automatically associate the **default** group to the new instance. Each project has a security group named **default** that provides default rules for the project. This **default** security group allows all IPv4 and IPv6 egress network traffic, and all IPv4 and IPv6 ingress traffic within the security group. This means that any instance within the **default** security group is open to any other instance in **default**. By default, any user in the project can edit the rules in that group.



NOTE

After you create an external network for the project, and you set the gateway for the router, an instance can actually get out to the external network. However, the services running inside the instance are not exposed to the outside world until you add a floating IP address to the instance and a security group with the proper rules.

Launching an Instance with Public Access

The following procedure outlines the steps necessary to launch an instance with public access.

1. Find the resources available for the instance. These may include an image, flavor, network, security group, and SSH key pair.
2. Launch the instance using the previous resources.
3. Ensure that the project network has a port on the router.
4. With an administrator account for the project, ensure that the external network exists and is set as the gateway for the router.
5. Ensure that there are available floating IP addresses allocated to the project. This can be done by a normal user.
6. Associate an available floating IP address to the instance.



REFERENCES

Further information is available in the Virtual Machine Instances section of the *Instances and Images Guide* for Red Hat OpenStack Platform at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/instances_and_images_guide/

LAUNCHING AN INSTANCE WITH PUBLIC ACCESS

In this exercise, you will launch an instance using resources created previously.

OUTCOMES

You should be able to launch an instance with public access using the resources created previously.

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

On **workstation**, run the **lab public-instance-deploy** script with the **setup** argument. It creates the required components for this exercise, such as users, passwords, images, and networks.

```
[student@workstation ~]$ lab public-instance-deploy setup
```

- ▶ 1. As the **developer1** user, delete all the existing instances in the **finance** project, if any.
 - 1.1. On **workstation**, source the **developer1** identity environment file.

```
[student@workstation ~]$ source ~/developer1-finance-rc  
[student@workstation ~(developer1-finance)]$
```

- 1.2. List all the instances in the project. This should be an empty list.

```
[student@workstation ~(developer1-finance)]$ openstack server list
```

- 1.3. If there are any instances running, delete them. Run the following command for each one.

```
[student@workstation ~(developer1-finance)]$ openstack server delete \  
instance-name
```

- ▶ 2. Launch an instance named **finance-server1** using the **rhe17** image, the **default** flavor, the **example-keypair** key pair, the **finance-network1** network, and the **default** security group.

```
[student@workstation ~(developer1-finance)]$ openstack server create \  
--image rhe17 \  
--flavor default \  
--key-name example-keypair \  
--nic net-id=finance-network1 \  
--security-group default \  
--wait finance-server1  
...output omitted...
```

- 3. As the project administrator (**architect1**), set the **provider-datacentre** network as the gateway for the **finance-router1** router.

3.1. Source the **architect1** identity environment file.

```
[student@workstation ~ (developer1-finance)]$ source ~/architect1-finance-rc  
[student@workstation ~ (architect1-finance)]$
```

3.2. Set the **provider-datacentre** network as the gateway for the router.

```
[student@workstation ~ (architect1-finance)]$ openstack router set \  
--external-gateway provider-datacentre finance-router1
```

- 4. As the **developer1** user, create three floating IP addresses for the project.

4.1. Source the **developer1** identity environment file.

```
[student@workstation ~ (architect1-finance)]$ source ~/developer1-finance-rc  
[student@workstation ~ (developer1-finance)]$
```

4.2. Create the first floating IP address.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip create \  
provider-datacentre  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| created_at | 2018-06-14T16:50:09Z |  
| description | |  
| fixed_ip_address | None |  
| floating_ip_address | 172.25.250.N |  
| floating_network_id | d2b4cff4-2444-4b65-ae40-c503a4f14980 |  
| id | a5fc67cc-8531-4d67-a6b7-e04c38d19132 |  
| name | 172.25.250.N |  
| port_id | None |  
| project_id | 438f276649894dcdbedb36ba1dc5296f |  
| qos_policy_id | None |  
| revision_number | 0 |  
| router_id | None |  
| status | DOWN |  
| subnet_id | None |  
| updated_at | 2018-06-14T16:50:09Z |  
+-----+-----+
```

4.3. Create the second floating IP address.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip create \  
provider-datacentre  
...output omitted...  
| floating_ip_address | 172.25.250.P |  
...output omitted...
```

4.4. Create the third floating IP address.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip create \  
provider-datacentre
```

```
provider-datacentre
...output omitted...
| floating_ip_address | 172.25.250.R
...output omitted...
```

- 5. Associate one of the floating IP addresses with the **finance-server1** instance and verify that the instance was assigned the floating IP address.

5.1. Associate one of the floating IP addresses with the **finance-server1** instance.

```
[student@workstation ~](developer1-finance)]$ openstack server add floating ip \
finance-server1 172.25.250.N
```

5.2. Verify that the instance was assigned the floating IP address.

```
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name \
-c Networks
+-----+-----+
| Name | Networks |
+-----+-----+
| finance-server1 | finance-network1=192.168.1.S, 172.25.250.N |
+-----+-----+
```

Cleanup

From **workstation**, run the **lab public-instance-deploy cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab public-instance-deploy cleanup
```

This concludes the guided exercise.

VERIFYING AN INSTANCE WITH PUBLIC ACCESS

OBJECTIVE

After completing this section, students should be able to verify an instance with public access.

HOW TO VERIFY AN EXTERNAL NETWORK

The procedure for verifying an instance with public access is very similar to verifying an external server: check the network and any services that are running. Verifying a connected network includes verifying that DHCP is providing IP addresses, DNS is resolving names, and network packets are traversing the proper routes. This usually involves checking the IP address assigned to the instance, checking that DNS names work, and pinging out from the instance. If static routes were created for the instance, these should be checked as well.

Pinging an Instance

After the instance has been assigned a floating IP address, one of the simplest networking tests is to ping the instance. The **ping** command sends an ICMP echo request, and expects an ICMP echo reply. If an echo reply is sent back from the instance, that indicates that the instance networking is functional. The following examples shows an ICMP echo request being sent to an instance using a floating IP address.

```
[user@demo ~]$ ping -c 3 -w 5 172.25.250.30
PING 172.25.250.30 (172.25.250.30) 56(84) bytes of data.
64 bytes from 172.25.250.30: icmp_seq=1 ttl=63 time=0.492 ms
64 bytes from 172.25.250.30: icmp_seq=2 ttl=63 time=0.492 ms
64 bytes from 172.25.250.30: icmp_seq=3 ttl=63 time=0.441 ms

--- 172.25.250.30 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.441/0.475/0.492/0.024 ms
```

Connecting via SSH

SSH can be used to connect to the instance. SSH provides a remote shell into the instance that allows a user to execute commands on the instance. The remote user account information on the instance must be known to the user accessing the shell. In some cases, that user account information is the user name and password contained in the image used to launch the instance. In most cases in the cloud, the user name is used with the SSH private key from the OpenStack key pair as credentials for the instance. The **root** account is usually prevented from logging in through SSH for security purposes. In Red Hat OpenStack Platform, the default user account created by **cloud-init** is the **cloud-user** account. This account is configured to use the SSH key pair specified when the instance launches. The **cloud-user** account also has full **sudo** privileges. The following example shows an SSH connection to an instance using a private key stored in `~/.ssh/keypair1.pem`.

```
[user@demo ~]$ ssh -i ~/.ssh/keypair1.pem cloud-user@172.25.250.30
Last login: Thu Dec 21 11:44:00 2017 from demo.lab.example.com
[cloud-user@demo-server1 ~]$
```



NOTE

SSH requires that the private key is not accessible to others, including its group permissions. A mode of **0600** (owner only has read and write permission) or **0400** (owner only has read permission) is recommended.

Verifying the External Network from the Instance

After logging in to the instance over SSH, ping out from the instance to verify external network connectivity. This tests the network connection out and in, as well as testing DNS name resolution. The following example shows a **ping** command as well as some DNS lookup commands from the instance.

```
[cloud-user@demo-server1 ~]$ ping -c 3 -w 5 redhat.com
PING redhat.com (209.132.183.105) 56(84) bytes of data.
64 bytes from redirect.redhat.com (209.132.183.105): icmp_seq=1 ttl=232 time=109
ms
64 bytes from redirect.redhat.com (209.132.183.105): icmp_seq=2 ttl=232 time=98.5
ms
64 bytes from redirect.redhat.com (209.132.183.105): icmp_seq=3 ttl=232 time=96.4
ms

--- redhat.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 96.401/101.476/109.480/5.738 ms
[cloud-user@demo-server1 ~]$ host redhat.com
redhat.com has address 209.132.183.105
redhat.com mail is handled by 10 mx2.redhat.com.
redhat.com mail is handled by 5 mx1.redhat.com.
[cloud-user@demo-server1 ~]$ dig redhat.com
; <>> DiG 9.9.4-RedHat-9.9.4-38.el7_3.2 <>> redhat.com
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 59309
;; flags: qr rd ra ad; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 0

;; QUESTION SECTION:
;redhat.com. IN A

;; ANSWER SECTION:
redhat.com. 3400 IN A 209.132.183.105

;; Query time: 0 msec
;; SERVER: 172.25.250.254#53(172.25.250.254)
;; WHEN: Tue Mar 28 09:29:21 EDT 2017
;; MSG SIZE rcvd: 44
```



NOTE

The **host** and **dig** commands, and many other useful DNS tools, are provided by the *bind-utils* package. The image used by the instance may not include that package. If that is the case, install the *bind-utils* package to provide these utilities.

Verifying Instance Services

After checking the instance networking, other services can be tested. For example, if the instance is running a web server, test a connection to the web server. If the instance is running a database server, attempt to access data from the server. The following example shows a test for HTTP and HTTPS using the instance's floating IP address.

```
[user@demo ~]$ curl http://172.25.250.30
My web site.
[user@demo ~]$ curl -k https://172.25.250.30
My web site.
```

TROUBLESHOOTING INSTANCE CONNECTIVITY

The following provides some guidelines when troubleshooting issues with instance connectivity.

Networking

Software-defined networking in OpenStack is an added layer of complexity when it comes to troubleshooting. Software-defined networks lay on top of the physical network layer, and this layer may need some troubleshooting when dealing with instances. When troubleshooting the network, view the network namespace using the **ip netns** command. Any command can be run within the namespace, although only the networking commands, such as **ping**, **host**, and **curl**, are useful within the namespace. The following list includes items that should be verified when troubleshooting OpenStack networking.

- Ensure that the instance was assigned the proper network. It is possible to create many private networks in OpenStack with the same name, and even using the same network range. Ensure that the correct network was used with the instance using the **openstack server list** command, which prints out the network names and IP addresses used by the instance.
- Ensure that packets can traverse through the OpenStack networking properly. Ping the instance from the network namespace on the network node on OpenStack. Ping the DHCP server and router IP addresses from the instance. Use commands such as **tracepath**, **traceroute**, or **mtr** to find how far network packets get.
- Ensure that the DHCP server is running and providing IP addresses. The **neutron_dhcp** container on the network node manages the DHCP servers associated with the project networks. Check that the instance got an IP address from the DHCP server. Use the **ip a** command to view the IP address for the instance. If the instance uses NetworkManager, the **/var/lib/NetworkManager/dhclient-*UUID*-eth0.lease** file contains the IP address of the DHCP server (the **option dhcp-server-identifier** line).
- Check routing with the **ip r** command to verify the correct routes.
- Verify DNS is working properly by using **host** or **dig** to validate host names.

SSH Key Pairs

SSH is an important part of accessing and administering any remote Linux system. In OpenStack, an SSH key pair can be assigned to an instance to provide access to the instance without the need for a password. This alleviates the need to embed an SSH public key within the image, and allows different SSH keys to be used with the same image.

SSH can be configured to lock out certain users, or to only allow certain users to SSH into the system. The **AllowUsers** and **AllowGroups** settings in **/etc/ssh/sshd_config** provide access to a list of users or groups. Similarly, the **DenyUsers** and **DenyGroups** denies access to a list of users or groups.

SSH can also be configured to only allow SSH keys instead of passwords for credentials.

PasswordAuthentication no set in **/etc/ssh/sshd_config** only allows SSH keys for authentication. Set the **PasswordAuthentication no** option in **/etc/ssh/sshd_config** to restrict authentication to SSH keys and to forbid the use of passwords even when a correct password is provided. When SSH keys are used for authentication, the public key must be placed in the authorized keys file (normally **~/.ssh/authorized_keys**). The private key can then be used for authentication. The private key should be kept private for security reasons. This includes changing the file permission to **0600** or **0400**, such that it is readable only by its owner.

When **cloud-init** injects the public key into the **cloud-user**'s **~/.ssh/authorized_keys** file, **cloud-init** also adds the SSH key to **root**'s **~/.ssh/authorized_keys** file with a subtle difference. The following code shows **root**'s **~/.ssh/authorized_keys** file.

```
[root@demo-server1 ~]$ cat /root/.ssh/authorized_keys
no-port-forwarding,no-agent-forwarding,no-X11-forwarding,command="echo 'Please
login as the user \"cloud-user\" rather than the user \"root\".';echo;sleep 10"
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQCb/LhEE0YjXgb+NYVsZV0uHQZNvJrwnYbInJqnUnw
XCHUgHq1ShtR7SxfR0Z6eW100PKTWjcNJU3ymDB1mTJTkZcgC61iddM4PsCVB9nKXiAnFrFBi9IsORT
jJHXDcukSR7nPIVNQy430W10i5ILo5iLdPP3PCjK1Zpr2z3VqG6wS3zTklJxCCAXd5o/6v5ksYfEz4Y
XKZmqzIMvbqqA12k+P1KAztq1V2I1urQ7CwjAD00m7VmS3BeQ3avcMkJtWDzgCUNhwBo/Tgesp2Zx+Z
2Pqm1dZ+JvoMnAv5a1EmE9ztgo9d5QRdBkE0LsyXicGvriF1SviRhTzc0h4GnYoX Generated-by-Nova
```

When attempting to log in as **root** using the SSH key, notice the output:

```
[user@demo ~]$ ssh -i ~/.ssh/keypair1.pem root@172.25.250.30
Please login as the user "cloud-user" rather than the user "root".
Connection to 172.25.250.30 closed.
```

Network Services

You troubleshoot network services exactly as you troubleshoot any other system. Each network service may be different, and troubleshooting all network services is beyond the scope of this book. However, troubleshooting a network service should include the following checks.

- Ensure that the correct packages are installed for the service. Ensure that the software repositories are available for these packages. If **cloud-init** is used to install packages, check the log files in the instance, including **/var/log/cloud-init.log** and **/var/log/yum.log**.
- Ensure that the service is started. If the image was specially built for the network service, ensure that the service was enabled at boot time. If **cloud-init** is used to manage the service, ensure that the service was enabled at boot time and started immediately. The **systemctl enable --now service-name** command can be used to both persistently enable a service and start a service immediately.
- Ensure that the configuration and other data is available to the service. Check permissions and privileges. Also verify that SELinux is allowing the service to access the data.

Security Group Rules

Any good system administrator knows to check firewall rules if there are issues connecting to a service that should be running on a system. In OpenStack, this is more complicated because there are several layers that a packet must pass through. First, the security group must pass the request through to the instance. Second, the firewall on the instance must pass the request through to the running service. For this reason, most images used in the cloud disable the firewall service entirely and rely on security groups to provide access.

Check if the instance is running a firewall by checking the **iptables** or **firewalld** service status. If one of those services is running, the ports must be open to allow access to services on the instance.

Security groups can be added to or removed from an instance after it has booted. Security group rules can be changed as well, so a user could add or remove a rule in the security group and all instances that include that security group would immediately allow or deny access.

Verifying an Instance with Public Access

The following procedure outlines the steps used to verify an instance with public access.

1. Find the instance and floating IP address using the **openstack server list** command.
2. Ping the instance from a system on (or outside) the external network.
3. Log in to the instance over SSH using the private SSH key. Use the **openstack server show *instance-name*** command to view the key pair name used by the instance.
4. From the instance, ping an external IP address. Check DNS name resolution.



REFERENCES

Further information is available in multiple sections of the *System Administrator's Guide* for Red Hat Enterprise Linux at
https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/system_administrators_guide/index

Further information is available in multiple sections of the *Networking Guide* for Red Hat Enterprise Linux at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

VERIFYING A INSTANCE WITH PUBLIC ACCESS

In this exercise, you will verify the functionality of an instance with public access by using **ssh** and **ping** to test the external network.



NOTE

This exercise uses the instance created in the previous exercise. If you have not completed the previous exercise, please do so now.

OUTCOMES

You should be able to verify that an instance with public access is properly functioning.

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

On **workstation**, run the **lab public-instance-verify** script with the **setup** argument. It ensures that the OpenStack services are running and the resources created previously are available to complete this exercise.

```
[student@workstation ~]$ lab public-instance-verify setup
```

- ▶ 1. Ping the instance, created previously as the **developer1** user, using the floating IP address.

1.1. On **workstation**, source the **developer1** identity environment file.

```
[student@workstation ~]$ source ~/developer1-finance-rc  
[student@workstation ~-(developer1-finance)]$
```

1.2. Find the floating IP address associated with the instance.

```
[student@workstation ~-(developer1-finance)]$ openstack server list \  
-c Name \  
-c Networks  
+-----+-----+  
| Name | Networks |  
+-----+-----+  
| finance-server1 | finance-network1=192.168.1.S, 172.25.250.N |  
+-----+-----+
```

1.3. Ping the instance.

```
[student@workstation ~-(developer1-finance)]$ ping -c 3 172.25.250.N  
PING 172.25.250.N (172.25.250.N) 56(84) bytes of data.  
64 bytes from 172.25.250.N: icmp_seq=1 ttl=64 time=0.365 ms
```

```
64 bytes from 172.25.250.N: icmp_seq=2 ttl=64 time=0.250 ms
64 bytes from 172.25.250.N: icmp_seq=3 ttl=64 time=0.239 ms

--- 172.25.250.N ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.239/0.284/0.365/0.060 ms
```

- 2. Log in to the instance as the **cloud-user** user, using the SSH key pair to authenticate.

```
[student@workstation ~]$ ssh -i /home/student/.ssh/example-keypair \
cloud-user@172.25.250.N
[cloud-user@finance-server1 ~]$
```

- 3. From the instance, ping **materials.example.com**. Exit from the instance.

- 3.1. From the instance, ping **materials.example.com**.

```
[cloud-user@finance-server1 ~]$ ping -c 3 materials.example.com
PING materials.example.com (172.25.254.254) 56(84) bytes of data.
64 bytes from classroom.example.com (172.25.254.254): icmp_seq=1 ttl=62 time=0.632 ms
64 bytes from classroom.example.com (172.25.254.254): icmp_seq=2 ttl=62 time=0.852 ms
64 bytes from classroom.example.com (172.25.254.254): icmp_seq=3 ttl=62 time=0.752 ms

--- materials.example.com ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.632/0.745/0.852/0.092 ms
```

- 3.2. Exit from the instance.

```
[cloud-user@finance-server1 ~]$ exit
logout
Connection to 172.25.250.N closed.
[student@workstation ~(developer1-finance)]$
```

Cleanup

From **workstation**, run the **lab public-instance-verify cleanup** script to clean up this exercise. The cleanup script deletes the instance.

```
[student@workstation ~]$ lab public-instance-verify cleanup
```

This concludes the guided exercise.

MANAGING MULTI-TENANT NETWORKING

OBJECTIVE

After completing this section, students should be able to manage multi-tenant networking.

OPENSTACK NETWORKING

Networking is one of the most complicated parts of OpenStack. The OpenStack networking service (neutron) provides software-defined networking (SDN). SDN provides the ability to define and manage networks and subnets in the software layer, without having to change hardware settings. Network namespaces allow these networks to be overlapping (for example, two networks using the same 192.168.0.0/24 subnet can be defined) without conflict. OpenStack networking also provides network functions virtualization (NFV). NFV is the virtualization of network functions, including routers and firewalls. Many parts of NFV (including orchestration) are beyond the scope of this course, but this section covers some of the functions of the software-defined router.

PROJECT NETWORKING

The following diagram provides a framework to describe the flow of network traffic between two instances in the same project, using the same network. The diagram shows the most complicated scenario of this flow—when the instances are on two different compute nodes.

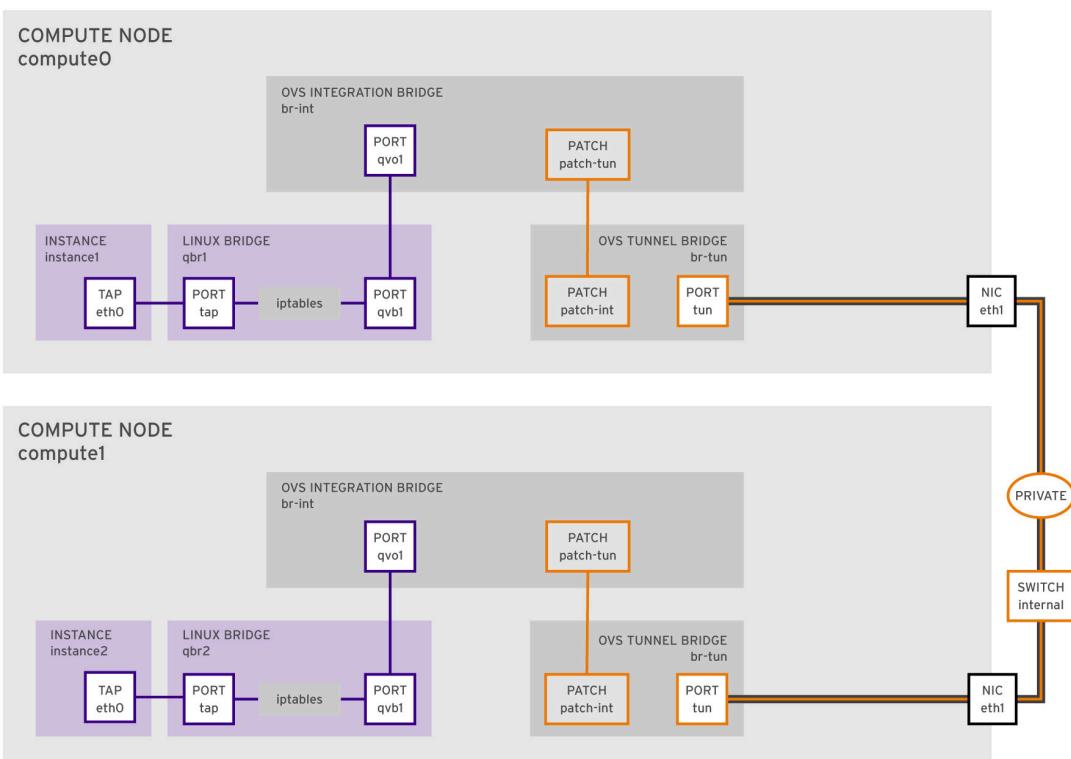


Figure 10.1: Networking in a single project, single network across two hosts

OpenStack Instance to Instance Traffic in a Single Project Network

The following flow describes the east-west path a network packet takes from one instance to another in the same project and network, across two compute hosts. Reference the previous diagram for the example interface, port, and bridge names.

1. On the **compute0** node, the **instance1** tap interface **eth0** forwards an outgoing packet to the Linux bridge **qbr1**. The packet contains the *destination* MAC address of the **instance2 eth0** interface, because they both are on the same logical segment. Security group rules for **instance1** provide packet state tracking and egress firewall rules. These rules are implemented as iptables entries on the Linux bridge.
2. The Linux bridge forwards the packet to the Open vSwitch (OVS) integration bridge **br-int**. The OVS integration bridge uses VLAN tagging to manage the virtual ports and networks connected to it.
3. The integration bridge forwards the packet to the OVS tunnel bridge **br-tun**. The tunnel bridge encapsulates the packet as a payload in a VXLAN Ethernet frame, adding a unique VXLAN Network Identifier (VNI) tag for this logical network, then sends the packet out the **eth1** interface to the **compute1** node.
4. On the **compute1** node, the tunnel interface **eth1** forwards the packet to the OVS tunnel bridge **br-tun**. The tunnel bridge removes the encapsulation of the packet and forwards it to the OVS integration bridge **br-int**.
5. The integration bridge uses the packet's *destination* MAC address to determine the correct local VLAN tag and connecting port, then forwards the packet to the Linux bridge **qbr2** on that port. Security group rules for **instance2** provide packet state tracking and ingress firewall rules.
6. The Linux bridge forwards the packet to the **instance2 eth0** interface through a tap device. Return traffic follows the same path back to **instance1** on **compute0**.



NOTE

Simple object names are used in these diagrams to help make them easy to read and understand. Real objects use globally unique IDs to create labels for namespaces, interfaces, ports, and devices. The following list shows some sample names found on running OpenStack nodes.

- namespace *qrouter-75697f67-4568-4261-b030-71d499c1d086*
- namespace *qdhcp-89b012f4-7693-43d4-a6a8-3233294b15ea*
- gateway port *qg-004086cc-18*
- router port *qr-48093af3-07*
- Linux bridge *qbr10ba76ca-16*
- veth pair Linux bridge endpoint *qvb10ba76ca-16*
- veth pair OVS bridge endpoint *qvo5dfb75e7-ad*
- TAP device *tap3736b551-9a*

The simpler scenario of the previous flow involves two instances on the same compute node. In that scenario, there is no need to send the packet through the tunnel bridge and the VXLAN tunnel. Everything else remains the same, including passing through the integration bridge. In the integration bridge, the packet is passed from the **qvo1** port to the **qvo2** port.

MULTIPROJECT NETWORKING

The following diagram provides a framework to describe the flow of network traffic from an instance to the external network, and between two instances in different projects using different networks.

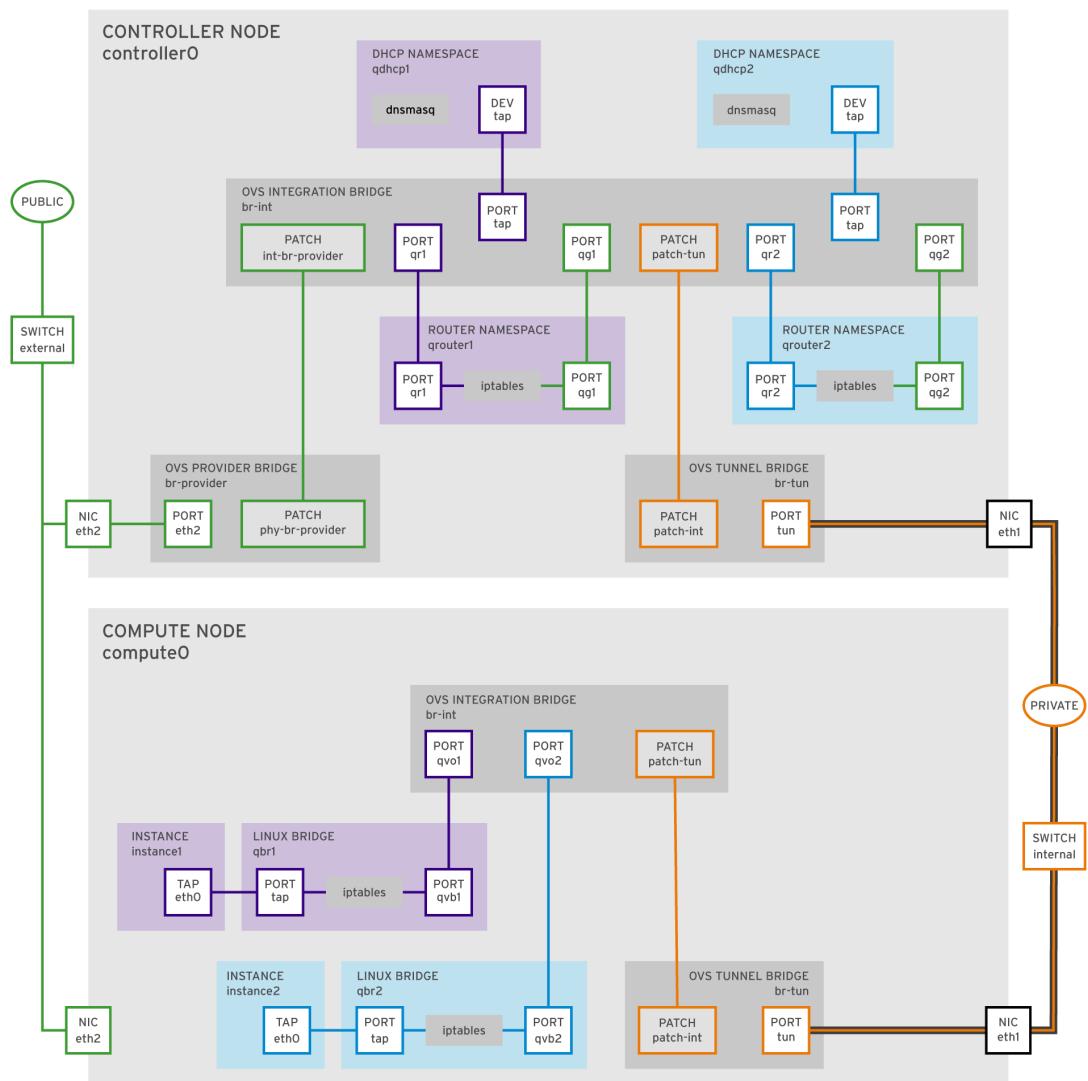


Figure 10.2: Networking between hosts in different projects

OpenStack Instance to External Networking Flow

The following flow describes the north-south path a network packet takes from an instance to the outside world.

1. On the *compute node*, the **instance1** tap interface (**eth0**) forwards the packet to the Linux bridge (**qbr1**). Security group rules for the instance provide packet state tracking and egress firewall rules.
2. The Linux bridge forwards the packet to the Open vSwitch (OVS) integration bridge (**br-int**). The OVS integration bridge adds an internal tag for the project network.
3. The integration bridge forwards the packet to the OVS tunnel bridge (**br-tun**). For the VXLAN project networks used in this course, the tunnel bridge encapsulates the packet as a payload in a VXLAN Ethernet frame, adds the internal tag for the project network, and sends the packet out of the compute node across the tunnel to the controller node.
4. On the *controller node*, the tunnel interface (**eth1**) forwards the packet to the OVS tunnel bridge (**br-tun**). The tunnel bridge unwraps the packet and adds the internal tag for the project network.
5. The tunnel bridge forwards the packet to the OVS integration bridge (**br-int**).

6. The integration bridge forwards the packet to the project router interface (**qr1**) in the router namespace (**qrouter1**). The **iptables** service performs a source network address translation, also known as SNAT, on the packet using the gateway IP address (on **qg1**) as the source IP address.
7. The router namespace forwards the packet back to the integration bridge (**br-int**) using the gateway interface (**qg1**).
8. The integration bridge forwards the packet to the OVS external bridge (**br-ex**).
9. The external bridge forwards the packet to the external network using the external interface (**eth0**).

The return traffic for the packet flows using similar steps in reverse.

OpenStack Instance to Instance Networking Flow (Different Projects)

The following flow describes the east-west path a network packet takes from an instance to another instance in a different project. The instances must have floating IP addresses to route network traffic between different projects.

1. On the *compute node*, the **instance1** tap interface (**eth0**) forwards the packet to the Linux bridge (**qbr1**). The packet contains the destination MAC address of the project 1 gateway interface (**qg1**) because the destination is on a different network. Security group rules for the instance provide packet state tracking and egress firewall rules.
2. The Linux bridge forwards the packet to the Open vSwitch (OVS) integration bridge (**br-int**). The OVS integration bridge adds an internal tag for the project 1 network.
3. The integration bridge forwards the packet to the OVS tunnel bridge (**br-tun**). For the VXLAN project networks used in this course, the tunnel bridge encapsulates the packet as a payload in a VXLAN Ethernet frame, adds the internal tag for the project 1 network, and sends the packet out of the compute node across the tunnel to the controller node.
4. On the *controller node*, the tunnel interface (**eth1**) forwards the packet to the OVS tunnel bridge (**br-tun**). The tunnel bridge unwraps the packet and adds the internal tag for the project 1 network.
5. The tunnel bridge forwards the packet to the OVS integration bridge (**br-int**).
6. The integration bridge forwards the packet to the router interface (**qr1**) in the project 1 router namespace (**qrouter1**). The **iptables** service performs SNAT on the packet using the gateway IP address as the source IP address.
7. The router namespace forwards the packet back to the OVS integration bridge (**br-int**) using the gateway interface (**qg1**).
8. The integration bridge forwards the packet to the OVS external bridge (**br-ex**).
9. The external bridge routes the packet back to the OVS integration bridge (**br-int**) since both floating IP addresses are on the same network.
10. The integration bridge forwards the packet to the gateway interface (**qg2**) in the project 2 router namespace (**qrouter2**), which has the **instance2** floating IP address. **iptables** performs DNAT on the packet to replace the floating IP as the destination IP address by the internal IP address of **instance2**.
11. The router namespace forwards the packet back to the OVS integration bridge (**br-int**) using the gateway interface (**qg2**). The integration bridge adds the internal tag for the project 2 network.

12. The integration bridge forwards the packet to the OVS tunnel bridge (**br-tun**). For the VXLAN project networks used in this course, the tunnel bridge encapsulates the packet as a payload in a VXLAN Ethernet frame, adds the internal tag for the project 2 network, and sends the packet out of the controller node across the tunnel to the compute node.
13. On the *compute node*, the tunnel interface (**eth1**) forwards the packet to the OVS tunnel bridge (**br-tun**). The tunnel bridge unwraps the packet and adds the internal tag for the project 2 network.
14. The tunnel bridge forwards the packet to the OVS integration bridge (**br-int**).
15. The integration bridge forwards the packet to the Linux bridge (**qbr2**) for project 2. Security group rules for **instance2** provide packet state tracking and ingress firewall rules.
16. The Linux bridge forwards the packet to the tap interface (**eth0**) on **instance2**.

The return traffic for the packet flows using similar steps in reverse.



NOTE

The previous diagram shows instances running on a single compute node, but the flow would be exactly the same if they were running on two different compute nodes.

Instance to Instance Flow in the Same Project

In the scenario where the network packet is traveling from one instance to another instance in the same project (and the same router namespace), the flow is a bit simpler. Instead of sending the packet out of the router namespace and to the external bridge (steps 7-10 in the previous flow), the router namespace routes the packet to the router interface (**qr2**) of the second network, and sends the packet to the tunnel bridge (over the integration bridge). The packet follows the same flow after that point as described previously.



REFERENCES

Further information is available in multiple sections of the *Networking Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/networking_guide/

Further information is available in the Deployment Scenarios section of the *Networking Guide* for OpenStack at
<https://docs.openstack.org/>

MANAGING MULTI-TENANT NETWORKING

The network path from an instance to the external network is shown as follows. Indicate the order the network path should flow.

1. The tunnel bridge forwards the packet to the integration bridge.
2. The integration bridge forwards the packet to the tunnel bridge. The tunnel bridge encapsulates the packet as a payload in a VXLAN Ethernet frame, adds the internal tag for the project network, and sends the packet out of the compute node across the tunnel to the controller node.
3. The Linux bridge forwards the packet to the integration bridge. The OVS integration bridge adds an internal tag for the project network.
4. The router namespace forwards the packet to the integration bridge using the gateway interface.
5. The external bridge forwards the packet to the external network using the external interface.
6. The instance tap interface forwards the packet to the Linux bridge. Security group rules for the instance provide packet state tracking and egress firewall rules.
7. The integration bridge forwards the packet to the router interface in the router namespace. The **iptables** service performs SNAT on the packet using the gateway IP address as the source IP address.
8. On the controller node, the tunnel interface forwards the packet to the tunnel bridge. The tunnel bridge unwraps the packet and adds the internal tag for the project network.
9. The integration bridge forwards the packet to the external bridge.

MANAGING MULTI-TENANT NETWORKING

The network path from an instance to the external network is shown as follows. Indicate the order the network path should flow.

- 5 1. The tunnel bridge forwards the packet to the integration bridge.
- 3 2. The integration bridge forwards the packet to the tunnel bridge. The tunnel bridge encapsulates the packet as a payload in a VXLAN Ethernet frame, adds the internal tag for the project network, and sends the packet out of the compute node across the tunnel to the controller node.
- 2 3. The Linux bridge forwards the packet to the integration bridge. The OVS integration bridge adds an internal tag for the project network.
- 7 4. The router namespace forwards the packet to the integration bridge using the gateway interface.
- 9 5. The external bridge forwards the packet to the external network using the external interface.
- 1 6. The instance tap interface forwards the packet to the Linux bridge. Security group rules for the instance provide packet state tracking and egress firewall rules.
- 6 7. The integration bridge forwards the packet to the router interface in the router namespace. The **iptables** service performs SNAT on the packet using the gateway IP address as the source IP address.
- 4 8. On the controller node, the tunnel interface forwards the packet to the tunnel bridge. The tunnel bridge unwraps the packet and adds the internal tag for the project network.
- 8 9. The integration bridge forwards the packet to the external bridge.

DEPLOYING AN INSTANCE WITH PUBLIC ACCESS

PERFORMANCE CHECKLIST

In this lab, you will launch an instance using resources created in the previous lab, and verify that the instance was properly launched.

OUTCOMES

You should be able to:

- Launch an instance with public access.
- Verify that an instance with public access is functioning properly.

Confirm the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab public-instance-lab** script with the **setup** argument. It creates the resources needed for this lab.

```
[student@workstation ~]$ lab public-instance-lab setup
```

1. From **workstation** as the **operator1** user, create an instance named **production-server1** using the following resources created in a previous chapter or by the lab **setup** script:

Instance Settings

TYPE	VALUE
Instance name	production-server1
Image	rhel7-web
Flavor	default
Key pair	production-keypair1
Project network	production-network1
Security group	production-secgroup1
Floating IP address	172.25.250.142

2. Associate the **172.25.250.142** floating IP address with the **production-server1** instance. Verify that you can SSH in to the instance using as **cloud-user** with the SSH private key in **/home/student/.ssh/production-keypair1**. From the instance, verify that you can ping the external server **materials.example.com**.

3. From **workstation**, verify that the instance responds to web server requests (HTTP and HTTPS).

Evaluation

From **workstation**, run the **lab public-instance-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab public-instance-lab grade
```

Cleanup

From **workstation**, run the **lab public-instance-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab public-instance-lab cleanup
```

This concludes the lab.

DEPLOYING AN INSTANCE WITH PUBLIC ACCESS

PERFORMANCE CHECKLIST

In this lab, you will launch an instance using resources created in the previous lab, and verify that the instance was properly launched.

OUTCOMES

You should be able to:

- Launch an instance with public access.
- Verify that an instance with public access is functioning properly.

Confirm the **workstation** and overcloud's virtual machines are started.

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

On **workstation**, run the **lab public-instance-lab** script with the **setup** argument. It creates the resources needed for this lab.

```
[student@workstation ~]$ lab public-instance-lab setup
```

1. From **workstation** as the **operator1** user, create an instance named **production-server1** using the following resources created in a previous chapter or by the lab **setup** script:

Instance Settings

TYPE	VALUE
Instance name	production-server1
Image	rhel7-web
Flavor	default
Key pair	production-keypair1
Project network	production-network1
Security group	production-secgroup1
Floating IP address	172.25.250.142

- 1.1. On **workstation**, open a terminal as **student** and source the **operator1** identity environment file **/home/student/operator1-production-rc**.

```
[student@workstation ~]$ source ~/operator1-production-rc  
[student@workstation ~(operator1-production)]$
```

- 1.2. Create the **production-server1** instance using the resources from the table.

```
[student@workstation ~(operator1-production)]$ openstack server create \  
--image rhe17-web \  
--flavor default \  
--key-name production-keypair1 \  
--nic net-id=production-network1 \  
--security-group production-secgroup1 \  
--wait \  
production-server1  
...output omitted...
```

2. Associate the **172.25.250.142** floating IP address with the **production-server1** instance. Verify that you can SSH in to the instance using as **cloud-user** with the SSH private key in **/home/student/.ssh/production-keypair1**. From the instance, verify that you can ping the external server **materials.example.com**.

- 2.1. Associate the floating IP address with the instance.

```
[student@workstation ~(operator1-production)]$ openstack server add \  
floating ip production-server1 172.25.250.142
```

- 2.2. Log in to the instance with SSH using the SSH private key.

```
[student@workstation ~(operator1-production)]$ ssh \  
-i ~/.ssh/production-keypair1 \  
cloud-user@172.25.250.142  
[cloud-user@production-server1 ~]$
```

- 2.3. From the instance, ping the **materials.example.com** server.

```
[cloud-user@production-server1 ~]$ ping -c 3 materials.example.com  
PING materials.example.com (172.25.254.254) 56(84) bytes of data.  
64 bytes from classroom.example.com (172.25.254.254): icmp_seq=1 ttl=62 time=0.851  
ms  
64 bytes from classroom.example.com (172.25.254.254): icmp_seq=2 ttl=62 time=0.589  
ms  
64 bytes from classroom.example.com (172.25.254.254): icmp_seq=3 ttl=62 time=0.839  
ms  
  
--- materials.example.com ping statistics ---  
3 packets transmitted, 3 received, 0% packet loss, time 2002ms  
rtt min/avg/max/mdev = 0.589/0.759/0.851/0.124 ms
```

- 2.4. Exit from the instance.

```
[cloud-user@production-server1 ~]$ exit  
[student@workstation ~(operator1-production)]$
```

3. From **workstation**, verify that the instance responds to web server requests (HTTP and HTTPS).

- 3.1. From **workstation**, verify that the instance responds to HTTP requests.

```
[student@workstation ~]$ curl http://172.25.250.142
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
...output omitted...
```

- 3.2. Verify that the instance responds to HTTPS requests.

```
[student@workstation ~]$ curl -k https://172.25.250.142
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN" "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" xml:lang="en">
<head>
  <title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>
...output omitted...
```

Evaluation

From **workstation**, run the **lab public-instance-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab public-instance-lab grade
```

Cleanup

From **workstation**, run the **lab public-instance-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab public-instance-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Launching an instance with public access is very similar to launching a private instance. Instances with public access often use key pairs and security groups for security. Verifying an instance is very similar to verifying an external server.
- The networking path between instances in the same project includes the Linux bridge and the integration bridge. If the instances are on different compute nodes, the networking path also includes the tunnel bridge and the VXLAN tunnel.
- The networking path from an instance to the external network includes the Linux bridge, the integration bridge, the network namespace, and the external bridge. If the compute node is separate from the controller node, networking path also includes the tunnel bridge and the VXLAN tunnel.
- The networking path between instances in different projects includes the Linux bridge, the integration bridge, the network namespaces, and the external bridge. If the instances are on different compute nodes or the compute node is separate from the controller node, the networking path also includes the tunnel bridge and the VXLAN tunnel multiple times.

CUSTOMIZING INSTANCES

GOAL

Customize an instance with cloud-init.

OBJECTIVES

- Customize an instance with cloud-init.
- Verify instance customization.

SECTIONS

- Creating Customized Instances (and Guided Exercise)
- Verifying Customized Instances (and Guided Exercise)

LAB

- Customizing Instances

CREATING CUSTOMIZED INSTANCES

OBJECTIVES

After completing this section, students should be able to customize an instance with cloud-init.

CLOUD-INIT

cloud-init is software that handles early initialization of an instance. It is included in the **rhel-guest-image** RPM, which is the base image provided by Red Hat. **cloud-init** can be used by administrators to perform tasks including:

- Setting a default locale, this can be dynamic rather than being baked into the image.
- Updating the instance host name.
- Generating or injecting SSH private keys to allow passwordless login.
- Setting up ephemeral mount points. Shared storage is a common requirement for horizontally scaled applications.

cloud-init can be invoked with *user-data*, which is data provided by the user when an instance is launched. The provided instructions are read and parsed by **cloud-init** in order to customize the instance. OpenStack also implements instance management via **cloud-init**. Users can launch an instance in the dashboard and use the Configuration tab to specify the customization to apply.

The screenshot shows the 'Launch Instance' dialog in the OpenStack dashboard. The 'Configuration' tab is selected. On the left, there's a sidebar with options like Details*, Source*, Flavor*, Networks*, Network Ports, Security Groups, Key Pair, Configuration (which is selected), Server Groups, Scheduler Hints, and Metadata. The 'Configuration' section contains a 'Customization Script' area with a large text input field. A note above it says: 'You can customize your instance after it has launched using the options available here. "Customization Script" is analogous to "User Data" in other systems.' Below the input field, it says 'Content size: 0 bytes of 16.00 KB'. There are 'Browse...' and 'No file selected.' buttons. At the bottom right of the dialog are 'Back', 'Next >', and a blue 'Launch Instance' button.

Figure 11.1: Customizing an instance while launching in the dashboard

OpenStack converts the information into a **cloud-init**-readable format. The following chart shows the user data flow initial configuration to the resulting instance customization.

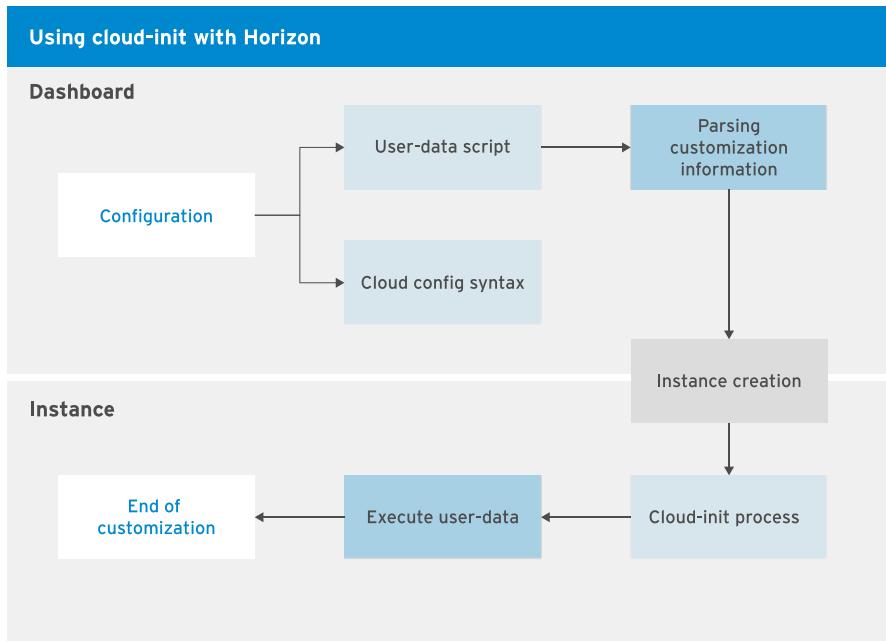


Figure 11.2: Customizing an instance using cloud-init flow chart

cloud-init DATA FORMATS

cloud-init contains support for several data formats which make it more flexible.

cloud-init has a maximum user data size of **16384 bytes** for data passed to an instance. Since the size limit cannot be changed, **gzip** compression is used to meet the size limit.

Administrators can use a MIME multi-part archive to contain more than one type of data. For example, both a **user-data** script and a **cloud-config** type can be included.

A **user-data** script that begins with **#!** or **Content-Type: text/x-shellscrip**t is supported. The script is executed at the **rc.local** level during the first boot of the instance. This script would run like any other script on the system.

cloud-init can include other files. The data declaration must begin with **#include** or **Content-Type: text/x-include-ur1**. The file contains a list of URLs, one per line. Each URL is read, and their contents are passed through this same set of rules, meaning that the content read from the URL can either be gzipped, MIME multipart, or plain text.

Data in the **cloud-config** format is supported. The data declaration must begin with **#cloud-config** or **Content-Type: text/cloud-config**.

Legacy run control **Upstart** jobs are also valid. The data declaration must begin with **#upstart-job** or **Content-Type: text/upstart-job**. The content is placed in a file under **/etc/init** used by upstart.

Data in the **Cloud Boothook** format is supported. The data declaration must begin with **#cloud-boothook** or **Content-Type: text/cloud-boothook**. The boot hook is the earliest execution time slot.

To extend cloud-init support for other MIME types, administrators can include a **part-handler**. The data declaration must begin with **#part-handler** or **Content-Type: text/part-handler**. The data is written to a file under **/var/lib/cloud/data** based on its file name. This handler must be written in Python.

user-data SCRIPTS

user-data scripts are a convenient way for administrators to send a set of instructions to the instance upon its creation. The script is invoked at the **rc.local** level, which is last in the boot process. The following example changes the message of the day, using a Bash script:

```
#!/bin/bash
echo "This instance has been customized by cloud-init at $(date -R)!" >> /etc/motd
```

cloud-config

As an alternative to using shell script, administrators can specify customization in **cloud-config** syntax, which provides instructions in a human-friendly format. These instructions include:

- Updating the system using **yum** on first boot which prevents the exposure of an instance that may not have the latest security updates installed.
- Adding a new yum repository allows access to different packages depending on the role of the instance.
- Importing SSH keys, which removes the requirement for password based logins and prevents brute-force attacks from succeeding.
- Creating users, which may be required for third-party agents such as backup or monitoring.



WARNING

The file must be valid YAML for it to be parsed and executed by **cloud-init**.

The following example shows how the system can be customized by adding a group that includes users, adding a new user, and running a set of commands.

```
#cloud-config
groups:
  - cloud-users: [john,doe]

users:
  - default
  - name: barfoo
    gecos: Bar B. Foo
    sudo: ALL=(ALL) NOPASSWD:ALL
    groups: users, admin
    ssh-import-id: None
    lock-passwd: true
    ssh-authorized-keys:
      - <SSH public key>

runcmd:
  - [ wget, "http://materials.example.com", -O, /tmp/index.html ]
  - [ sh, -xc, "echo $(date) ': hello world!'" ]
```

Creating Customized Instances (Dashboard)

The following steps outline the process to launch an instance from the dashboard and customize it.

1. Log into **workstation** as **student** using **student** as the password. From **workstation** run **demo customize-instances setup**, which ensures OpenStack services are running and that the required OpenStack resources exist.
2. Open the dashboard URL in a web browser, and navigate to Project → Compute → Instances.
3. Click **Launch Instance**.
4. Populate the fields in the Details, Source, Flavor and Networks tabs. Optionally, make changes in the Security Groups and Key Pair tabs.
5. In the Configuration tab, provide your customization commands directly in the Configuration Script field, or upload a script file.
6. Click **Launch Instance**.
7. When the instance is marked as **Active**, attach a floating IP address to it.
8. Open a terminal and log into the instance using the **ssh** command. Connect to the floating IP address using an appropriate private key.
9. Query the status of the **cloud-init** service, to ensure it is enabled and running.

Creating Customized Instances (command line)

The following steps outline the process to launch a customized instance from the command line.

1. Open a terminal and source an identity environment file.
2. Create a file with instructions for **cloud-init**. The instructions may be in any format cloud-init supports, such as a simple shell script.
3. Launch an instance and pass **--user-data /path_to/your_file** as an argument.
4. When the instance state is **active**, attach a floating IP address.
5. Open a terminal and log into the instance with **ssh** using the floating IP address and the appropriate private key.
6. Query the status of the **cloud-init** service, to ensure it is enabled and running.



REFERENCES

Further information on cloud-init is available at
<https://cloudinit.readthedocs.io>

CREATING CUSTOMIZED INSTANCES

In this exercise, you will customize two instances using **cloud-init** capabilities and features. You will log into the first instance to confirm **cloud-init** is up and running.

OUTCOMES

You should be able to:

- Customize an instance using a **cloud-config** script.
- Customize an instance using a **cloud-config** file.

Log in to **workstation** as **student** using **student** as the password. From **workstation**, run **lab customize-instance setup**, which ensures OpenStack services are running and that the required OpenStack resources exist.

```
[student@workstation ~]$ lab customize-instance setup
```

- 1. Customize an instance using the dashboard. Log in to the Horizon dashboard as **developer1** using **redhat** as the password. Launch an instance named **finance-server1** with the **rhe17** image, the **default** flavor, the **finance-network1** network, the **default** security group, and the **example-keypair** key pair. Create a customization script that will include "Hello world!" in the **/root/hello.txt** file in the instance.
- 1.1. From **workstation**, open a web browser and navigate to `http://dashboard.overcloud.example.com`. Log in using **developer1** as the user, and **redhat** as the password.
 - 1.2. Navigate to Compute → Instances.
 - 1.3. Click Launch Instance.
 - 1.4. In the Details tab, enter **finance-server1** as the Instance Name.
 - 1.5. In the Source tab, choose Image in the Select Boot Source menu. Select the image **rhe17** by clicking on the up arrow on the same row, in the Available section. Click No under Create New Volume.
 - 1.6. In the Flavor tab, click the up arrow on the **default** row.
 - 1.7. In the Networks tab, click up arrow on the **finance-network1** row.
 - 1.8. In the Security Groups tab, ensure that the **default** security group has been selected.
 - 1.9. In the Key Pair tab, ensure that the **example-keypair** key pair has been selected.
 - 1.10. In the Configuration tab, populate the Customization Script field with the following content:

```
#!/bin/sh
echo 'Hello world!' > /root/hello.txt
```

- 1.11. Click Launch Instance.

- ▶ 2. After the instance is active, attach a floating IP address to the **finance-server1** instance.

- 2.1. Once the instance Status is **Active**, attach a floating IP address to it. Select Associate Floating IP from the Actions menu in the row for the instance.
Create a new floating IP address by clicking on the + button in the IP Address section. Click on Allocate IP, then click on Associate.
- 2.2. Log out of the dashboard.

- ▶ 3. On **workstation**, create a **user-data** file, called **install_httpd**, to customize the instance. The script will install the web server and enable the service.

```
[student@workstation ~ (developer1-finance)]$ vim /home/student/install_httpd
#!/bin/bash
# web server with a postgres backend
yum -y install httpd python-psycopg2
systemctl enable httpd --now
```

- ▶ 4. Customize an instance using the command line. Use the **developer1** user to launch an instance named **finance-server2** with the **rhel7** image, the **default** flavor, the **finance-network1** network, the **default** security group, and the **example-keypair** key pair. Include the user-data script **/home/student/install_httpd**.

- 4.1. Launch an instance using the **--user-data** option to perform the customization.

```
[student@workstation ~ (developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--nic net-id=finance-network1 \
--security-group default \
--key-name example-keypair \
--user-data /home/student/install_httpd \
--wait finance-server2
```

- 4.2. Verify that the status of the instance **finance-server2** is **active**.

```
[student@workstation ~ (developer1-finance)]$ openstack server list \
-c Name -c Status -c Networks
+-----+-----+-----+
| Name | Status | Networks |
+-----+-----+-----+
| finance-server2 | ACTIVE | finance-network1=192.168.1.11 |
| finance-server1 | ACTIVE | finance-network1=192.168.1.7, 172.25.250.104 |
+-----+-----+-----+
```

- ▶ 5. Once the instance state is **active**, create a new floating IP address and attach it to the instance. Verify that the floating IP has been assigned.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip create \
provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| created_at | 2018-06-25T06:13:50Z |
| description | |
```

```
| fixed_ip_address | None           |
| floating_ip_address | 172.25.250.YY |
| floating_network_id | 95a56df3-a710-4bc6-b7c5-fc5d3e725df6 |
| id                 | 57d2d3b2-f206-45e3-ad9c-14fecefabd79 |
| name               | 172.25.250.YY |
| port_id            | None           |
| project_id         | e6b7f37fadf943e1aae06a5c2e55d646 |
| qos_policy_id      | None           |
| revision_number    | 0              |
| router_id          | None           |
| status              | DOWN           |
| subnet_id          | None           |
| updated_at          | 2018-06-25T06:13:50Z |
+-----+
[student@workstation ~](developer1-finance)]$ openstack server add floating ip \
finance-server2 172.25.250.YY
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name -c Networks
+-----+
| Name       | Networks           |
+-----+
| finance-server2 | finance-network1=192.168.1.11, 172.25.250.YY |
| finance-server1 | finance-network1=192.168.1.7, 172.25.250.NN |
+-----+
```

Cleanup

From **workstation**, run the command **lab customize-instance cleanup** to clean up this exercise.

```
[student@workstation ~]$ lab customize-instance cleanup
```

This concludes the guided exercise.

VERIFYING CUSTOMIZED INSTANCES

OBJECTIVES

After completing this section, students should be able to verify instance customization.

VERIFYING INSTANCE CUSTOMIZATION

Once instances have been spawned, administrators can ensure the **cloud-init** instructions were successfully executed. Customization can include:

- Installing a package.
- Removing a package.
- Updating the system.
- Creating users or groups.
- Retrieving a file.

To confirm the correct operation of **cloud-init**, check the **cloud-init** log file. **cloud-init** logs its operations into the log file **/var/log/cloud-init.log**. To confirm the customization, check the system for the expected results.

For example, check **/var/log/cloud-init.log** to confirm that **cloud-init** ran:

```
[user@demo ~]$ sudo less /var/log/cloud-init.log
...
Jun 19 01:53:05 host-192-168-1-5 cloud-init: Cloud-init v. 0.7.6 finished at Tue,
19 Jun 2018 06:53:05 +0000. Datasource DataSourceOpenStack [net,ver=2]. Up 88.48
seconds
```

In the **/etc/passwd** you can check if the **nagios** user was created:

```
[user@demo ~]$ grep nagios /etc/passwd
...
nagios:x:903:903::/home/nagios:/bin/bash
```

Using the **ps -ef** you can ensure that services are running.

```
[user@demo ~]$ ps -ef | grep httpd
root      1204      1  0 04:38 ?          00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    1205  1204  0 04:38 ?          00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    1206  1204  0 04:38 ?          00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    1207  1204  0 04:38 ?          00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    1208  1204  0 04:38 ?          00:00:00 /usr/sbin/httpd -DFOREGROUND
apache    1209  1204  0 04:38 ?          00:00:00 /usr/sbin/httpd -DFOREGROUND
cloud-u+  1258  1237  0 04:40 pts/0    00:00:00 grep --color=auto httpd
```

Using the **systemctl** you can ensure that services are active and enabled.

```
[user@demo ~]$ systemctl status httpd.service
● httpd.service - The Apache HTTP Server
  Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset: disabled)
  Active: active (running) since Tue 2018-06-19 04:38:38 EDT; 2min 26s ago
    Docs: man:httpd(8)
          man:apachectl(8)
  Main PID: 1204 (httpd)
    Status: "Total requests: 0; Current requests/sec: 0; Current traffic: 0 B/sec"
   CGroup: /system.slice/httpd.service
           ├─1204 /usr/sbin/httpd -DFOREGROUND
           ├─1205 /usr/sbin/httpd -DFOREGROUND
           ├─1206 /usr/sbin/httpd -DFOREGROUND
           ├─1207 /usr/sbin/httpd -DFOREGROUND
           ├─1208 /usr/sbin/httpd -DFOREGROUND
           └─1209 /usr/sbin/httpd -DFOREGROUND
...output omitted
```

Verifying Instance Customization

The following steps outline the process to verify that **cloud-init** has correctly customized an instance.

1. Determine the floating IP address allocated to the instance.
2. Log in to the instance with the floating IP address using **ssh** and an appropriate private key.
3. Check for the results of any customization, for example, whether a package is installed, or a service is running.
4. Review the **/var/log/cloud-init.log** file for relevant messages.



REFERENCES

Further information on cloud-init is available at
<https://cloudinit.readthedocs.io>

VERIFY CUSTOMIZED INSTANCES

In this exercise, you will verify that **cloud-init** has correctly customized the two instances created in the previous section.

OUTCOMES

You should be able to verify **cloud-init** operations by checking the **/var/log/cloud-init.log** file, or by confirming that the requested customization has occurred.

Ensure that the **workstation** and overcloud virtual machines are started.

Log in to **workstation** as **student** using **student** as the password. From **workstation**, run the **lab customization-instance-verify** script with the **setup** argument which ensures that the instances created during the previous section are still running.

```
[student@workstation ~]$ lab customize-instance-verify setup
```

- ▶ 1. From **workstation**, open a terminal, source the **developer1-finance-rc** environment file, then determine the floating IP address allocated to the **finance-server1** instance. Remember that the floating IP address is in the **172.25.250.0/24** subnet.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack server list \
-c Name -c Networks
+-----+-----+
| Name | Networks |
+-----+-----+
| finance-server2 | finance-network1=192.168.1.NN, 172.25.250.YY |
| finance-server1 | finance-network1=192.168.1.MM, 172.25.250.XX |
+-----+-----+
```

In the book, you will see the floating IP address associated with the **finance-server1** instance written as **172.25.250.XX**. Replace the last octet (**XX**) with the last octet in the output above.

- ▶ 2. Log in to **finance-server1** to verify that the **cloud-init** customization script created the **/root/hello.txt** file.
- 2.1. Log in to **finance-server1** with **ssh** using the **example-keypair** private key.

```
[student@workstation ~(developer1-finance)]$ ssh -i ~/.ssh/example-keypair \
cloud-user@172.25.250.XX
```

2.2. Check **/var/log/cloud-init.log** to confirm that **cloud-init** ran.

```
[cloud-user@finance-server1 ~]$ sudo less /var/log/cloud-init.log
...output omitted...
```

```
...util.py[DEBUG]: Cloud-init v. 0.7.9 finished at Sat, 23 Jun 2018 02:26:02  
+0000. Datasource DataSourceOpenStack [net,ver=2]. Up 21.25 seconds
```

2.3. Ensure that the **/root/hello.txt** file exists and has the correct content.

```
[cloud-user@finance-server3 ~]$ sudo cat /root/hello.txt  
Hello world!
```

2.4. Log out from **finance-server1**.

```
[cloud-user@finance-server1 ~]$ exit  
logout  
Connection to 172.25.250.XX closed.
```

► 3. Log in to **finance-server2** to verify that the **/home/student/install_httpd** user-data script, installed and enabled the **httpd** service.

3.1. Determine the floating IP address allocated to the **finance-server2** instance.

```
[student@workstation ~-(developer1-finance)]$ openstack server show finance-server2  
\  
-c addresses -f value  
finance-server2=192.168.1.MM, 172.25.250.YY
```

In the book, you will see the floating IP address associated with the **finance-server2** instance written as 172.25.250.YY. Replace the last octet (YY) with the last octet in the output above.

3.2. Using **ssh** and the **example-keypair** private key, log in to **finance-server2**.

```
[student@workstation ~-(developer1-finance)]$ ssh -i ~/.ssh/example-keypair \  
cloud-user@172.25.250.YY
```

3.3. Check **/var/log/cloud-init.log** to confirm that **cloud-init** ran.

```
[cloud-user@finance-server2 ~]$ sudo less /var/log/cloud-init.log  
...output omitted...  
...util.py[DEBUG]: Cloud-init v. 0.7.9 finished at Sat, 23 Jun 2018 02:26:02  
+0000. Datasource DataSourceOpenStack [net,ver=2]. Up 88.48 seconds
```

3.4. Confirm that **httpd** is working.

```
[cloud-user@finance-server2 ~]$ curl http://localhost | grep Test  
<title>Test Page for the Apache HTTP Server on Red Hat Enterprise Linux</title>  
<h1>Red Hat Enterprise Linux <strong>Test Page</strong></h1>
```

3.5. On **workstation**, use the **curl** to navigate to **http://172.25.250.YY**. The connection must succeed.

```
[student@workstation ~-(developer1-finance)]$ curl http://172.25.250.YY  
...output omitted...
```

Cleanup

On **workstation**, run the command **lab customize-instance-verify cleanup** to clean up the exercise.

```
[student@workstation ~]$ lab customize-instance-verify cleanup
```

This concludes the guided exercise.

CUSTOMIZING INSTANCES

PERFORMANCE CHECKLIST

In this lab, you will run a script on an instance, add a file to an instance, then verify your work.

OUTCOMES

You should be able to install a package or file on an instance using **cloud-init**, then verify your work using the cloud-init log file.

Log in to **workstation** as **student** using **student** as a password.

From **workstation**, run **lab customize-instance-lab setup**, which ensures OpenStack services are running and that the required OpenStack resources exist.

```
[student@workstation ~]$ lab customize-instance-lab setup
```

1. Create a file with instructions for **cloud-init** to install the **mariadb-server** package. The instructions should also start and enable the **mariadb** service.
2. Launch an instance named **production-server1** using the **rhel7** image, **default** flavor, **production-network1** network, **example-keypair** key pair, **default** security group, and the user-data file **install_mariadb**. Create a floating ip address in the **provider-datacentre** network. Assign the created floating ip address to the **production-server1** instance.
3. Using **ssh** and the **example-keypair** key pair log in to **production-server1**.
4. Verify that **cloud-init** ran successfully.

Evaluation

On **workstation**, run the **lab customize-instance-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab customize-instance-lab grade
```

Cleanup

On **workstation**, run the **lab customize-instance-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab customize-instance-lab cleanup
```

This concludes the lab.

CUSTOMIZING INSTANCES

PERFORMANCE CHECKLIST

In this lab, you will run a script on an instance, add a file to an instance, then verify your work.

OUTCOMES

You should be able to install a package or file on an instance using **cloud-init**, then verify your work using the cloud-init log file.

Log in to **workstation** as **student** using **student** as a password.

From **workstation**, run **lab customize-instance-lab setup**, which ensures OpenStack services are running and that the required OpenStack resources exist.

```
[student@workstation ~]$ lab customize-instance-lab setup
```

1. Create a file with instructions for **cloud-init** to install the **mariadb-server** package. The instructions should also start and enable the **mariadb** service.
 - 1.1. Source the **operator1-production-rc** environment file. Create an instruction file in the **/home/student** directory called **install_mariadb** with the following instructions.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ 
[student@workstation ~(operator1-production)]$ vim /home/student/install_mariadb
#!/bin/bash

yum -y install mariadb-server
systemctl enable mariadb --now
```

2. Launch an instance named **production-server1** using the **rhe17** image, **default** flavor, **production-network1** network, **example-keypair** key pair, **default** security group, and the user-data file **install_mariadb**. Create a floating ip address in the **provider-datacentre** network. Assign the created floating ip address to the **production-server1** instance.
 - 2.1. Launch the **production-server1** instance.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--image rhe17 \
--nic net-id=production-network1 \
--key-name example-keypair \
--user-data /home/student/install_mariadb \
--wait production-server1
+-----+
| Field           | Value          |
+-----+
```

OS-DCF:diskConfig	MANUAL
OS-EXT-AZ:availability_zone	nova
OS-EXT-STS:power_state	Running
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
OS-SRV-USG:launched_at	2018-06-20T07:24:34.000000
OS-SRV-USG:terminated_at	None
accessIPv4	
accessIPv6	
addresses	production-network1=192.168.1.9
adminPass	7SyNrmSpURSD
config_drive	
created	2018-06-20T07:24:15Z
flavor	default (a2db...064c)
hostId	159f...4955
id	17d0...840d
image	rhel7 (2078...25a2)
key_name	example-keypair
name	production-server1
progress	0
project_id	adcf...d074
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-20T07:24:34Z
user_id	0f80...4b82
volumes_attached	

2.2. Create the floating ip address in the **provider-datacentre** network.

[student@workstation ~ (operator1-production)]\$ openstack floating ip create \ provider-datacentre	
Field	Value
created_at	2018-06-20T07:31:08Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.XX
floating_network_id	6d845ef5-8844-490e-a944-cbd22a453c78
id	d4c9ebcc-c443-4fb7-8ecd-e5ec446c311d
name	172.25.250.111
port_id	None
project_id	adcf826c851145638d2f49e9fe56d074
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-20T07:31:08Z

2.3. Attach the floating ip address to the **production-server1** instance.

[student@workstation ~ (operator1-production)]\$ openstack server add \

```
floating ip production-server1 172.25.250.XX  
[student@workstation ~]$(operator1-production)]$
```

- 2.4. Ensure that the floating ip has been assigned to **production-server1**

```
[student@workstation ~]$(operator1-production)]$ openstack server list \  
-c Name -c Networks  
+-----+-----+  
| Name | Networks |  
+-----+-----+  
| production-server1 | production-network1=192.168.1.9, 172.25.250.XX |  
+-----+-----+
```

3. Using **ssh** and the **example-keypair** key pair log in to **production-server1**.

- 3.1. Verify the **production-server1** instance is active.

```
[student@workstation ~]$(operator1-production)]$ openstack server show \  
production-server1  
...output omitted...  
| OS-EXT-STS:vm_state | active  
...output omitted...
```

- 3.2. Log in to the **production-server1** instance as the **cloud-user** using the **example-keypair** private key found at **~/.ssh/example-keypair** and the floating IP address.

```
[student@workstation ~]$(operator1-production)]$ ssh -i ~/.ssh/example-keypair \  
cloud-user@172.25.250.XX  
Warning: Permanently added '172.25.250.111' (ECDSA) to the list of known hosts.  
[cloud-user@production-server1 ~]$
```

4. Verify that **cloud-init** ran successfully.

- 4.1. Check **/var/log/cloud-init.log** to confirm that **cloud-init** ran successfully.

```
[cloud-user@production-server1 ~]$ sudo less /var/log/cloud-init.log  
...output omitted...  
2018-06-23 00:11:40,287 - util.py[DEBUG]: Cloud-init v. 0.7.9 finished at Sat, 23  
Jun 2018 04:11:40 +0000. Datasource DataSourceOpenStack [net,ver=2]. Up 46.99  
seconds
```

- 4.2. Confirm that mariadb is installed and running.

```
[cloud-user@production-server1 ~]$ rpm -q mariadb-server  
mariadb-server-10.1.20-2.el7ost.x86_64  
[cloud-user@production-server1 ~]$ systemctl status mariadb  
● mariadb.service - MariaDB database server  
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; vendor  
   preset: disabled)  
     Active: active (running) since Tue 2018-06-19 18:29:50 EST; 2min 53s ago  
       Process: 1196 ExecStartPost=/usr/libexec/mariadb-wait-ready $MAINPID  
   (code=exited, status=0/SUCCESS)  
     Process: 1116 ExecStartPre=/usr/libexec/mariadb-prepare-db-dir %n (code=exited,  
   status=0/SUCCESS)
```

```
Main PID: 1195 (mysqld_safe)
CGroup: /system.slice/mariadb.service
    ├─1195 /bin/sh /usr/bin/mysqld_safe --basedir=/usr
    └─1359 /usr/libexec/mysqld --basedir=/usr --datadir=/var/lib/mysql --
plugin-dir=/usr/lib64/mysql/plugin --log-error=/var/log/mariadb...
...output omitted...
```

Evaluation

On **workstation**, run the **lab customize-instance-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab customize-instance-lab grade
```

Cleanup

On **workstation**, run the **lab customize-instance-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab customize-instance-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- **cloud-init** is the tool used by OpenStack to perform instance customization. It allows users to customize their instances without the overhead of integrating with a configuration management system.
- **cloud-init** supports several data formats, including cloud-config and shell script. Existing scripts can be reused directly, reducing the effort required to align cloud instances with existing standards. The cloud-config format can be simpler to maintain than shell script but requires some effort to recreate existing customization.
- Users provide data to **cloud-init** with the dashboard or with the command-line interface.

DEPLOYING SCALABLE STACKS

GOAL

Deploy a stack and configure Auto Scaling.

OBJECTIVES

- Analyze OpenStack metrics for use in Auto Scaling.
- Deploy a stack.
- Configure stack Auto Scaling.

SECTIONS

- Analyzing Cloud Metrics for Auto Scaling (and Guided Exercise)
- Deploying a Stack (and Guided Exercise)
- Configuring Stack Auto Scaling (and Guided Exercise)

LAB

- Deploying Scalable Stacks

ANALYZING CLOUD METRICS FOR AUTO SCALING

OBJECTIVES

After completing this section, students should be able to describe the architecture of Telemetry service, Alarming service, and agent plug-ins.

TELEMETRY ARCHITECTURE AND SERVICES

In Red Hat OpenStack Platform the Telemetry service provides user-level usage data for OpenStack components. The Telemetry service provides metric data to support billing systems for OpenStack cloud resources. The Telemetry service gathers information about the system and stores it to provide data required for billing purposes. This data can be fed into cloud management software, such as Red Hat CloudForms™, to provide itemized billing and a chargeback to the cloud users.

The Telemetry service collects data using polling agents and notification agents. Polling agents poll the OpenStack infrastructure resources, such as the hypervisor, to publish meters on the notification bus. The notification agent listens to the notifications on the OpenStack notification bus and converts them into meter events and samples. Most OpenStack resources are able to send such events using the notification system built into *oslo.messaging*. The normalized data collected by the Telemetry service is then published to various targets.

Sample data collected by various agents is published to the storage targets. The **openstack metric** command allows executing query requests on this data store by authenticated users. Query requests on a data store return a list of resources and statistics based on various metrics collected.

A timeseries database code named **gnocchi** was introduced to decouple the storing of metric data and to increase efficiency. This time series database service is often referred to as the metric service. The Alarming service (aodh) manage alarms created by a user to monitor an OpenStack resource.

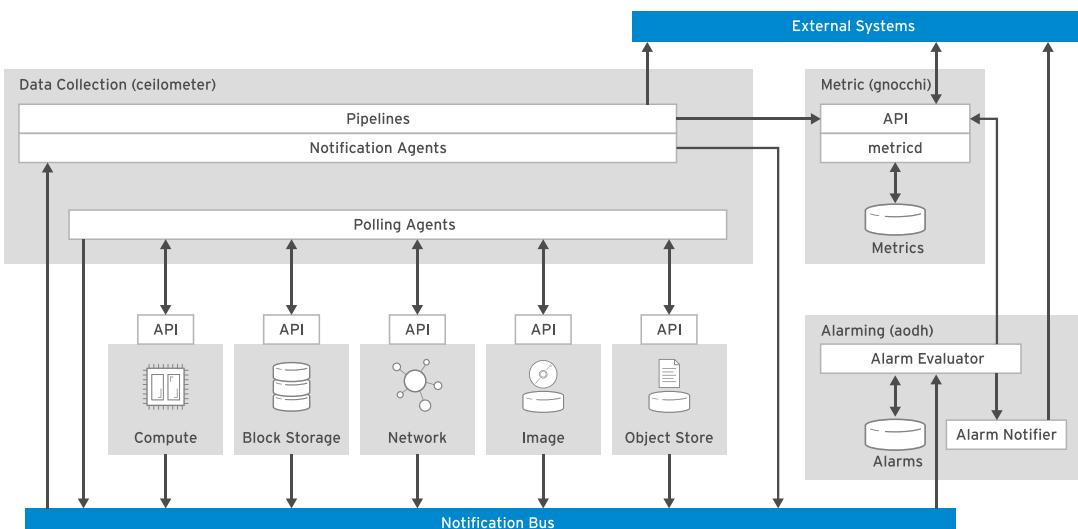


Figure 12.1: Telemetry service components

Metering and Data Collection Service

The data collection service (ceilometer) collects data by using two built-in plug-ins:

- Notification agents: This is the preferred method for collecting data. An agent monitors the message bus for data sent by different OpenStack services such as Compute, Image, Block Storage, Orchestration, Identity, and so on. Messages are then processed by various plugins to convert them into events and samples.
- Polling agents: These agents poll services for collecting data. Polling agents are either configured to get information about the hypervisor or using a remote API such as IPMI to gather the power state of a compute node. This method is not optimal, as it increases the load on the Telemetry service API endpoint.

Data gathered by notification and polling agents are processed by various transformers to generate data samples. For example, to get a CPU utilization percentage, multiple CPU utilization sample data collected over a period can be aggregated. Processed data samples get published to the time series database for long term storage or to an external system using a publisher.

Polling Agent Plug-ins

The Telemetry service uses a polling agent to gather information about the infrastructure that is not published by events and notifications from OpenStack components. The polling agents use the APIs exposed by the different OpenStack services and other hardware assets such as compute nodes. The Telemetry service uses agent plugins to support this polling mechanism. The three default agents plugins used for polling are:

- Compute agent: This agent gathers resource data about all instances running on different compute nodes. The compute agent is installed on every compute node to facilitate interaction with the local hypervisor. Sample data collected by a compute agent is sent to the message bus. The sample data is processed by the notification agent and published to different publishers.
- Central agent: These agents use the REST APIs of various OpenStack services to gather additional information that was not sent as a notification. A central agent polls networking, object storage, block storage, and hardware resources using SNMP. The sample data collected is sent to the message bus to be processed by the notification agent.
- IPMI agent: This agent uses the **ipmitool** utility to gather IPMI sensor data. An IPMI-capable host requires that an IPMI agent is installed. The sample data gathered is used for providing metrics associated with the physical hardware.

Timeseries Database Service

The metric service (gnocchi) is a time series database used to store metrics and resources published by the Telemetry service. A time series database is optimized for handling data that contains arrays of numbers indexed by time stamp. The metric service provides a REST API to create or edit metric data, as well as insight on cloud usage, billing, placement, chargeback, and capacity planning. The **gnocchi-metricd** service computes statistics, in real time, on received data. This computed data is stored and indexed for fast retrieval.

The time series database supports various back ends for storing the metric data and indexed data. Currently supported storage drivers for storing metric data include file, Ceph, Swift, S3, and Redis. The time series database uses PostgreSQL or MySQL database to store indexed data and any associated metadata. The default storage driver for indexed data is *PostgreSQL*.

Gnocchi Architecture

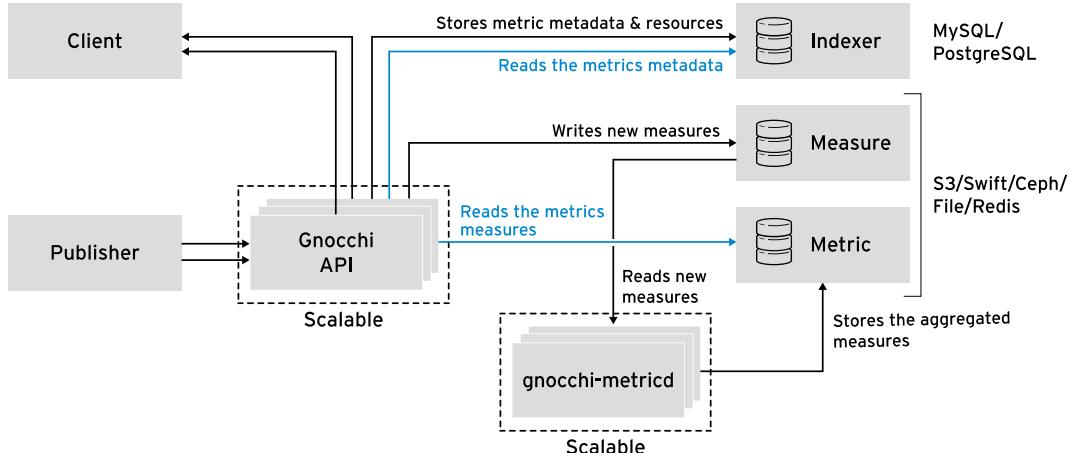


Figure 12.2: Time Series Database Architecture

The data collection service uses the **Gnocchi API** service to publish data samples to the metric service for processing and storage. Received data samples are stored in temporary **measure** data store. The **gnocchi-metricd** service reads the measures from the **measure** data store. The **gnocchi-metricd** service then computes the measures based on the archive policy and the aggregation methods defined for the meter. The computed statistics are then stored for long term in the **metric** data store.

To retrieve the metric data, a client (such as the Alarming service) uses the **Gnocchi API** service to read the metric measures from the metric storage, and the metric metadata stored in the index storage.

Alarming Service

The Alarming service (aodh) provides the alarming services within the Telemetry architecture. For example, you might want to trigger an alarm when CPU utilization of an instance reaches 70% for more than 10 minutes. To create an alarm, the alarm action and conditions need to be defined.

An alarm rule is used to define when the alarm is to be triggered. The alarm rule can be based on an event or on a computed statistic. The definition of an action to be taken when the alarm is triggered supports multiple forms:

- An HTTP callback URL, invoked when the alarm is triggered.
- A log file to log the event information.
- A notification sent using the messaging bus.

RETRIEVE AND ANALYZE OPENSTACK METRICS

The Telemetry service stores the metrics associated with various OpenStack services persistently using the Time Series Database (gnocchi) service. An authenticated user is allowed to send a request to a Time Series Database service API endpoint to read the measures stored in the data store.

Time Series Database Resources

Resources are objects that represent cloud components, such as an instance, volume, image, load balancer VIP, host, IPMI sensor, and so on. The measures stored in the Gnocchi Time Series Database service are indexed based on the resource and its attributes.

Time Series Database Measure

A *measure* in the Time Series Database service is the data gathered for a resource at a given time. The Time Series Database service stores the measure, which is a lightweight component; each measure includes a number, a time stamp, and a value.

```
[user@demo ~(user-demo)]$ openstack metric measures show \
--resource-id a509ba1e-91df-405c-b966-c41b722dfd8d \
cpu_util
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2017-06-14T00:00:00+00:00 | 86400.0 | 0.542669194306 |
| 2017-06-14T15:00:00+00:00 | 3600.0 | 0.542669194306 |
| 2017-06-14T15:40:00+00:00 | 300.0 | 0.542669194306 |
+-----+-----+-----+
```

Time Series Database Metrics

The Time Series Database service provides an entity called *metric* that stores the aspect of the resource in the data store. For example, if the resource is an instance, the aspect is the CPU utilization, which is stored as a metric. Each metric has several attributes: a metric ID, a name, an archive policy that defines storage lifespan, and different aggregates of the measures.

```
[user@demo ~(user-demo)]$ openstack metric show \
--resource-id 6bd6e073-4e97-4a48-92e4-d37cb365cddb \
image.serve
+-----+
| Field | Value
+-----+
| archive_policy/aggregation_methods | std, count, 95pct, min, max, sum, median, mean |
| archive_policy/back_window | 0
| archive_policy/definition | - points: 12, granularity: 0:05:00, timespan: |
| | | 1:00:00
| | |
| | | - points: 24, granularity: 1:00:00, timespan: |
| | | 1 day, 0:00:00 | 1:00:00
| | | - points: 30, granularity: 1 day, 0:00:00, timespan: 30 days, 0:00:00 |
| | |
| | | - points: 30, granularity: 1 day, 0:00:00, timespan: 30 days, 0:00:00 |
| archive_policy/name | low
| ...
| ...output omitted...
| id | 2a8329f3-8378-49f6-aa58-d1c5d37d9b62
| |
| name | image.serve
| |
```

Time Series Database Archive Policy

The archive policy defined by an OpenStack administrator specifies the data storage policy in the Time Series Database service. For example, an administrator can define a policy to store data for one day with one second granularity, or one hour granularity of data to be stored for one year, or both. Aggregation methods, such as min, max, mean, sum, and so on, provided by the Time Series Database service are used to aggregate the measures based on granularity specified in the policy. The aggregated data is stored in the database according to the archive policies. Archive policies are defined on a per-metric basis and are used to determine the lifespan of stored aggregated data.

Three archive policies are defined by default: **low**, **medium**, and **high**. The archive policy to be used depends on your use case. Depending on the usage of the data, you can either use one of the default policies or define your own archive policy.

Time Series Database default archive policies

POLICY NAME	ARCHIVE POLICY DEFINITION
low	<ul style="list-style-type: none">Stores metric data with 5 minutes granularity over 30 days.
medium	<ul style="list-style-type: none">Stores metric data with 1 minute granularity over 7 days.Stores metric data with 1 hour granularity over 365 days.
high	<ul style="list-style-type: none">Stores metric data with 1 second granularity over one hour.Stores metric data with 1 minute granularity over 7 days.Stores metric data with 1 hour granularity over 365 days.

Using the OpenStack CLI to Analyze Metrics

The command-line tool provided by the *python-gnocchiclient* package helps retrieve and analyze the metrics stored in the Time Series Database service. The **openstack metric** command is used to retrieve and analyze the Telemetry metrics.

To retrieve all the resources and the respective resource IDs, use the **openstack metric resource list** command.

```
[user@demo ~(user-demo)]$ openstack metric resource list -c type -c id
+-----+-----+
| id | type |
+-----+-----+
| 4464b986-4bd8-48a2-a014-835506692317 | image |
| 05a6a936-4a4c-5d1b-b355-2fd6e2e47647 | instance_disk |
| cef757c0-6137-5905-9edc-ce9c4d2b9003 | instance_network_interface |
| 6776f92f-0706-54d8-94a1-2dd8d2397825 | instance_disk |
| dbf53681-540f-5ee1-9b00-c06bb53cbd62 | instance_disk |
| cebc8e2f-3c8f-45a1-8f71-6f03f017c623 | swift_account |
| a2b3bda7-1d9e-4ad0-99fe-b4f7774deda0 | instance |
+-----+-----+
```

To list the metrics associated with a resource, use the **openstack metric resource show** command. The resource ID is retrieved using the **openstack metric resource list --type** command, which filters based on resource type.

```
[user@demo ~(user-demo)]$ openstack metric resource list --type image -c type -c id
+-----+-----+
| id | type |
+-----+-----+
| 4464b986-4bd8-48a2-a014-835506692317 | image |
+-----+-----+
[user@demo ~(user-demo)]$ openstack metric resource show 4464b986-4bd8-48a2-a014-835506692317
+-----+
| Field | Value |
+-----+
| created_by_project_id | d42393f674a9488abe11bd0ef6d18a18 |
| created_by_user_id | 7521059a98cc4d579eea897027027575 |
| ended_at | None |
| id | 4464b986-4bd8-48a2-a014-835506692317 |
| metrics | image.download: 7b52afb7-3b25-4722-8028-3d3cc3041316 |
| | image.serve: 2e0027b9-bc99-425f-931a-a3afad313cb3 |
| | image.size: ff4b7310-e6f9-4871-98b8-fff2006fb897 |
| | image: b0feed69-078b-4ab7-9f58-d18b293c110e |
| original_resource_id | 4464b986-4bd8-48a2-a014-835506692317 |
| project_id | fd0ce487ea074bc0ace047accb3163da |
| revision_end | None |
| revision_start | 2017-05-16T03:48:57.218470+00:00 |
| started_at | 2017-05-16T03:48:57.218458+00:00 |
| type | image |
| user_id | None |
+-----+
```

All the metrics provided by the Telemetry service can be listed by an OpenStack administrator using the **openstack metric list** command.

```
[user@demo ~(admin)]$ openstack metric list -c name -c unit -c archive_policy/name
+-----+-----+-----+
| archive_policy/name | name | unit |
+-----+-----+-----+
| low | disk.iops | None |
| low | disk.root.size | GB |
| low | subnet.create | None |
| low | storage.objects.outgoing.bytes | None |
| low | disk.allocation | B |
| low | network.update | None |
| low | disk.latency | None |
| low | disk.read.bytes | B |
...output omitted...
```

The **openstack metric metric show** command shows the metric details. The resource ID of a resource is retrieved using the **openstack metric resource list** command.

To list detailed information of the **image.serve** metric for an image with the **6bd6e073-4e97-4a48-92e4-d37cb365cddb** resource ID, run the following command:

```
[user@demo ~ (user-demo)]$ openstack metric show \
--resource-id 74df9ee2-0881-4654-97b3-caca2f0b08d9 \
```

Field	Value
archive_policy/name	low
creator	ef6cde08413746d6b0a1dea1a8565522:553a5434e6
	3842eb80513c9e740b61e7
id	8766b800-c7bb-4196-ba6c-49d2b5cee061
name	image.serve
resource/created_by_project_id	553a5434e63842eb80513c9e740b61e7
resource/created_by_user_id	ef6cde08413746d6b0a1dea1a8565522
resource/creator	ef6cde08413746d6b0a1dea1a8565522:553a5434e6
	3842eb80513c9e740b61e7
resource/ended_at	2018-07-04T04:53:24.170752+00:00
resource/id	74df9ee2-0881-4654-97b3-caca2f0b08d9
resource/original_resource_id	74df9ee2-0881-4654-97b3-caca2f0b08d9
resource/project_id	3b328fd576c84f098df8a605e71bdf59
resource/revision_end	None
resource/revision_start	2018-07-04T04:53:24.993289+00:00
resource/started_at	2018-07-04T04:41:49.233310+00:00
resource/type	image
resource/user_id	None
unit	None

The **openstack metric archive-policy list** command lists the archive policies.

[user@demo ~(user-demo)]\$ openstack metric archive-policy list -c name -c definition	
name	definition
bool	- points: 31536000, granularity: 0:00:01, timespan: 365 days, 0:00:00
high	- points: 3600, granularity: 0:00:01, timespan: 1:00:00
	- points: 10080, granularity: 0:01:00, timespan: 7 days, 0:00:00
	- points: 8760, granularity: 1:00:00, timespan: 365 days, 0:00:00
low	- points: 8640, granularity: 0:05:00, timespan: 30 days, 0:00:00
medium	- points: 10080, granularity: 0:01:00, timespan: 7 days, 0:00:00
	- points: 8760, granularity: 1:00:00, timespan: 365 days, 0:00:00
[user@demo ~(user-demo)]\$ openstack metric archive-policy list -c name -c aggregation_methods	
name	aggregation_methods
bool	last
high	std, count, min, max, sum, mean
low	std, count, min, max, sum, mean
medium	std, count, min, max, sum, mean

A Telemetry service administrator can add measures to the data store using the **openstack metric measures add** command. To view measures, use **openstack metric measures show**. Both commands require the metric name and resource ID as parameters.

The Time Series Database service uses ISO 8601 time stamp format for output. In ISO 8601 notation, the date, time, and time zone are represented in the following format: **yyyymmddThhmmss+|-hhmm**. The **date -u --iso=seconds** command converts the current date time into the ISO 8601 timestamp format.

Measures are added using the **yyyymmddThhmmss+|-hhmm@value** format. Multiple measures can be added using the **openstack metric measures add --measure** command.

The resource ID of a resource is retrieved using the **openstack metric resource list** command. To list the metrics associated with a resource, use the **openstack metric resource show** command. The default aggregation method used by the **openstack metric resource show** command is **mean**.



IMPORTANT

For removing measures, administrator privileges are required.

The final entry in the following output shows the *average* CPU utilization for a resource.

```
[user@demo ~(admin)]$ openstack metric measures add \
--resource-id a509ba1e-91df-405c-b966-c41b722dfd8d \
--measure $(date -u --iso=seconds)@23 \
cpu_util
[user@demo ~(admin)]$ openstack metric measures show \
--resource-id a509ba1e-91df-405c-b966-c41b722dfd8d \
--refresh cpu_util
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2017-04-27T00:00:00+00:00 | 86400.0 | 11.0085787895 |
| 2017-04-27T11:00:00+00:00 | 3600.0 | 0.312042039086 |
| 2017-04-27T12:00:00+00:00 | 3600.0 | 16.3568471647 |
| 2017-04-27T11:45:00+00:00 | 300.0 | 0.374260637142 |
| 2017-04-27T11:55:00+00:00 | 300.0 | 0.24982344103 |
| 2017-04-27T12:05:00+00:00 | 300.0 | 0.263997402134 |
| 2017-04-27T12:15:00+00:00 | 300.0 | 0.163391256752 |
| 2017-04-27T12:20:00+00:00 | 300.0 | 32.5 |
+-----+-----+-----+
```



NOTE

For querying and adding measures, a few other time stamp formats are supported. For example: **50 minutes**, which indicating 50 minutes from now, and **- 50 minutes**, indicating 50 minutes ago. Time stamps based on the UNIX epoch are also supported.

Use aggregation methods such as **min**, **max**, **mean**, **sum**, and so on, to display the measures based on the granularity.

The following command shows how to list measures with a particular aggregation method. The command uses the resource ID associated with an instance to display the minimum CPU utilization for different granularity. The **--refresh** option is used to include all new measures. The final entry of the following screen capture shows the *minimum* CPU utilization for the resource.

```
[user@demo ~(user-demo)]$ openstack metric measures show \
--resource-id a509ba1e-91df-405c-b966-c41b722dfd8d \
--aggregation min \
--refresh cpu_util
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2017-04-27T00:00:00+00:00 | 86400.0 | 0.163391256752 |
| 2017-04-27T11:00:00+00:00 | 3600.0 | 0.24982344103 |
| 2017-04-27T12:00:00+00:00 | 3600.0 | 0.163391256752 |
| 2017-04-27T11:45:00+00:00 | 300.0 | 0.374260637142 |
| 2017-04-27T11:55:00+00:00 | 300.0 | 0.24982344103 |
| 2017-04-27T12:05:00+00:00 | 300.0 | 0.263997402134 |
| 2017-04-27T12:15:00+00:00 | 300.0 | 0.163391256752 |
| 2017-04-27T12:20:00+00:00 | 300.0 | 23.0 |
+-----+-----+-----+
```

Querying the Telemetry Metrics

An administrator can query the Telemetry metrics stored in the database based on several conditions. For example, you can specify a time range to look for measures based on the aggregation method. Operators like equal to (**eq**), less than or equal to (**le**), greater than or equal to (**ge**), lesser than (**lt**), greater than (**gt**), and various not operators can be used in the query. The operators **or**, **and**, and **not** are also supported.

The **--query** option uses attributes associated with a resource type. The following command displays the mean CPU utilization for all provisioned instances that use the flavor with an ID of **1**, or that use the image with an ID of **6bd6e073-4e97-4a48-92e4-d37cb365cddb**.

```
[user@demo ~(admin)]$ openstack metric resource-type show instance
+-----+
+-----+
| Field | Value
|       |
+-----+
+-----+
| attributes/display_name | max_length=255, min_length=0, required=True,
| type=string |
| attributes/flavor_id | max_length=255, min_length=0, required=True,
| type=string |
| attributes/host | max_length=255, min_length=0, required=True,
| type=string |
| attributes/image_ref | max_length=255, min_length=0, required=False,
| type=string |
| attributes/server_group | max_length=255, min_length=0, required=False,
| type=string |
| name | instance
|     |
| state | active
|     |
+-----+
+-----+
[user@demo ~(admin)]$ openstack metric measures \
aggregation \
--metric cpu_util \
```

```
--aggregation mean \
--resource-type instance \
--query '(flavor_id=1)or(image_ref='6bd6e073-4e97-4a48-92e4-d37cb365cddb')' \
--refresh
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2017-06-14T00:00:00+00:00 | 86400.0 | 0.575362745414 |
...output omitted...
```

Use the **--start** option and the **--stop** option in the **openstack metric measures aggregation** command to provide the time range for computing aggregation statistics. For example, the **server_group** attribute of the instance resource type can be used the **--query** option to group a specific set of instances which can then be monitored for auto-scaling. It is also possible to search for values in the metrics by using one or more levels of granularity. Use the **--granularity** option to make queries based on the granularity.

```
[user@demo ~ (admin)]$ openstack metric measures \
aggregation \
--metric cpu_util \
--aggregation mean \
--resource-type instance \
--start $(date -u --iso=seconds -d "-60mins") \
--granularity 60 \
--query "server_group='finance'" \
--needed-overlap 0 \
--refresh
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2018-07-04T14:32:00+05:30 | 60.0 | 0.1858290391 |
| 2018-07-04T14:42:00+05:30 | 60.0 | 0.1401828896 |
...output omitted...
```

Common Telemetry Metrics

The Telemetry service collects different meters by polling the infrastructural components or by consuming notifications provided by various OpenStack services. There are three types of metric data provided by the Telemetry service:

- Cumulative: A cumulative meter provides measures that are accumulated over time. For example, total CPU time used.
- Gauge: A gauge meter records the current value at the time that a reading is recorded. For example, number of images.
- Delta: A delta meter records the change between values recorded over a particular time period. For example, network bandwidth.

Some meters collected for instances are:

Compute service meters

METER NAME	METER TYPE	DESCRIPTION
memory	gauge	Amount of memory in MB allocated to the instance

METER NAME	METER TYPE	DESCRIPTION
memory.usage	gauge	Amount of memory in MB consumed by the instance
cpu	cumulative	Total CPU time used in nanosecond (ns)
cpu.util	gauge	Average CPU utilization in percentage
disk.read.requests	cumulative	Total number of read requests
disk.read.requests.rate	gauge	Average rate of read requests per second

Meters collected for images are:

Image service meters

METER NAME	METER TYPE	DESCRIPTION
image	gauge	Size of the image uploaded
image.download	delta	Number of bytes downloaded for an image
image.serve	delta	Number of bytes served out for an image

TELEMETRY METRICS IN AUTO SCALING

When using Auto Scaling to scale in and scale out of instances, the Telemetry service alerts provided by the alarm managed by the Alarming service trigger execution of the Auto Scaling policy. The alarm watches a single metric published by the metric service, and alarms send messages to Auto Scaling when the metric crosses the threshold value. The alarm can monitor any metrics provided by the metric service. Most common metrics for auto-scaling an instance are **cpu.util**, **memory.usage**, **disk.read.requests.rate**, and **disk.write.requests.rate**. However, custom metrics can also be used to trigger auto-scaling.

Monitoring OpenStack Resources with the Telemetry Service

The following steps outline the process for monitoring cloud resources using the Telemetry service:

1. Use the **openstack metric resource list** command to find the resource ID and the desired resource.
2. Use the **openstack metric resource show** command with the resource ID found in the previous step to view the available meters for the resource. Make note of the metric ID.
3. Use the **openstack metric show** command with the metric ID found in the previous step to view the details of the desired meter.
4. Use the **openstack metric measures show** command with the metric ID found in the previous step to list the measures.
5. Use the **openstack metric measures aggregation** command with the metric name to list the computed aggregated measures.



REFERENCES

Gnocchi Project Architecture

<http://gnocchi.xyz/intro.html>

For more information, refer to the *Configuring the Time Series Database (Gnocchi) for Telemetry* chapter in the *Logging, Monitoring, and Troubleshooting Guide* for Red Hat OpenStack Platform at

https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/logging_monitoring_and_troubleshooting_guide/configuring_the_time_series_database_gnocchi_for_telemetry

ANALYZING CLOUD METRICS FOR AUTO SCALING

In this exercise, you will view and analyze common metrics required for auto-scaling.

OUTCOMES

You should be able to:

- List available metrics associated with a resource.
- Analyze metrics to view aggregated values.

Ensure that the **workstation** and overcloud's virtual machines are started.

On **workstation**, run the **lab stacks-metrics setup** command. This script ensures that the environment is properly configured for this exercise. The script also creates an instance named **finance-server1**.

```
[student@workstation ~]$ lab stacks-metrics setup
```

- ▶ 1. From **workstation**, source the **/home/student/developer1-finance-rc** file. List the resource types available in the Telemetry service. List the metrics associated with the **image** resource type.
- 1.1. List the resource types available.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack metric resource-type \
list -c name
+-----+
| name
+-----+
| ceph_account
| generic
| host
| host_disk
| host_network_interface
| identity
| image
| instance
| instance_disk
| instance_network_interface
| ipmi
| manila_share
| network
| nova_compute
| port
| stack
```

```
| swift_account          |
| switch                 |
| switch_port            |
| switch_table           |
| volume                 |
| volume_provider         |
| volume_provider_pool   |
+-----+
```

- 1.2. List the resources accessible to the **developer1** user. Note the resource ID of the **image** resource type.

```
[student@workstation ~ (developer1-finance)]$ openstack metric resource list \
-c id -c type
+-----+-----+
| id             | type      |
+-----+-----+
| b9f90067-0c8c-4144-81de-8b0fa55b791c | image     |
| 05a6a936-4a4c-5d1b-b355-2fd6e2e47647 | instance_disk |
| cef757c0-6137-5905-9edc-ce9c4d2b9003 | instance_network_interface |
| 6776f92f-0706-54d8-94a1-2dd8d2397825 | instance_disk |
| dbf53681-540f-5ee1-9b00-c06bb53cbd62 | instance_disk |
| cebc8e2f-3c8f-45a1-8f71-6f03f017c623 | swift_account |
| a2b3bda7-1d9e-4ad0-99fe-b4f7774deda0 | instance    |
| ...           | ...       |
+-----+-----+
```

- 1.3. Verify that the ID of the **rhel7** image is the same as the resource ID.

```
[student@workstation ~ (developer1-finance)]$ openstack image list \
-c ID -c Name
+-----+-----+
| ID           | Name      |
+-----+-----+
| b9f90067-0c8c-4144-81de-8b0fa55b791c | rhel7     |
| ...         | ...       |
+-----+-----+
```

- 1.4. List the metrics associated with the **image** resource ID.

```
[student@workstation ~ (developer1-finance)]$ openstack metric resource show \
b9f90067-0c8c-4144-81de-8b0fa55b791c
+-----+-----+
| Field          | Value      |
+-----+-----+
| created_by_project_id | 553a5434e63842eb80513c9e740b61e7 |
| created_by_user_id | ef6cde08413746d6b0a1dea1a8565522 |
| creator         | ef6cde08413746d6b0a1dea1a8565522:553a5434e63842eb805 |
|                | 13c9e740b61e7 |
| ended_at        | None       |
| id              | b9f90067-0c8c-4144-81de-8b0fa55b791c |
| metrics         | image.download: 19c10cb6-dd57-43aa-8bce-e3a1ee4c9039 |
|                | image.serve: 6317f1e6-ce0c-41fc-b8c9-266beddc2738 |
|                | image.size: 149da247-1911-4073-a296-77350324606f |
| original_resource_id | b9f90067-0c8c-4144-81de-8b0fa55b791c |
+-----+-----+
```

project_id	3b328fd576c84f098df8a605e71bdf59
revision_end	None
revision_start	2018-07-02T09:34:15.213043+00:00
started_at	2018-07-02T09:34:15.213029+00:00
type	image
user_id	None

- 2. Using the appropriate resource ID, list all metrics associated with the **finance-server1** instance. List the details for the **disk.read.requests.rate** metric.

2.1. List all metrics associated with the **finance-server1** instance.

```
[student@workstation ~(developer1-finance)]$ openstack server list -c ID -c Name
+-----+-----+
| ID | Name |
+-----+-----+
| 8e8ca3d6-6b1c-4a97-a242-ec8549f6628e | finance-server1 |
+-----+-----+
[student@workstation ~(developer1-finance)]$ openstack metric resource show \
8e8ca3d6-6b1c-4a97-a242-ec8549f6628e -f json
{
    "created_by_user_id": "ef6cde08413746d6b0a1dea1a8565522",
    "user_id": "630d039328fb44b79c97685d335b833f",
    "started_at": "2018-07-04T10:23:43.925033+00:00",
    "ended_at": null,
    "revision_end": null,
    "creator": "41fa2761227c4722b2676baa9cd451cc:553a5434e63842eb80513c9e740b61e7",
    "created_by_project_id": "553a5434e63842eb80513c9e740b61e7",
    "metrics": {
        "cpu_l3_cache": "608d3cab-01d8-4c2a-aeec-dc7bbd449c60",
        "disk.write.requests.rate": "3dbd5011-7dfc-44d3-a13d-8d3fb8f0cc45",
        "perf.cache.misses": "6fed6e38-b2ef-455e-973d-ea7a1b382116",
        "perf.cache.references": "c9ff224b-512a-42e4-9ab7-e26cbfae53d6",
        "disk.latency": "993bab86-e5dd-44b8-8539-9dd450eeee260",
        "disk.ephemeral.size": "7aa3a359-2bd6-47ef-9df0-5e70cdde9d8d",
        "disk.read.requests": "5126c317-32d3-4484-803a-8ae1c387e74c",
        "perf.cpu.cycles": "cd95c496-f9ea-4ead-9e7c-8ffe9cdc1776",
        "disk.write.bytes": "e63186be-e043-4d1c-a5a1-49225b4a83a0",
        "memory.resident": "4451aec5-93ba-4e72-bb61-ba0c40e683e4",
        "disk.capacity": "6d3a7c24-4fee-47ae-bd13-ed0a197b1800",
        "disk.root.size": "0757cc68-0e90-4820-99c3-96a3377fcc10",
        "disk.usage": "d6e607b2-6889-47c4-af4b-6b0686f07957",
        "disk.read.bytes.rate": "0d2e1264-a82d-4f8c-9663-0896d12e5baf",
        "cpu_util": "9de294d4-42da-464f-bdfa-4724852e2192",
        "memory.bandwidth.total": "51e5a704-92b5-4e00-adbb-60e48b571a21",
        "memory": "f2c2c55d-b646-42af-b194-d3f9fa2c5364",
        "disk.write.bytes.rate": "82ad14e3-5de2-4bc8-840b-ab791c9dc5ac",
        "memory.usage": "87222d08-7607-4f49-9ad8-a63cc8ddf82a",
        "memory.swap.in": "50c506a4-dc6b-41e0-9def-8cb2590c2f92",
        "cpu.delta": "6367665e-f578-4173-8a0a-97920dc3ae3a",
        "compute.instance.booting.time": "ac442b38-53a0-4d03-92e3-b9fb348192b8",
        "perf.instructions": "d741dabf-91d5-4a0a-84a3-9a302f3d25b8",
        "disk.read.requests.rate": "74678be7-239c-455d-8eba-da6ea9ea7392",
        "disk.iops": "b7b13aac-a62e-4083-9c1a-3a20e215af9c",
        "memory.bandwidth.local": "e36a12c7-00f0-4fba-bc95-5fc28046ed94"
    }
}
```

```

    "disk.allocation": "b080148e-091d-40f2-b1b9-3f15fabad69e",
    "disk.write.requests": "8189e670-7f79-4389-abac-f0b7750f5dc9",
    "vcpus": "9b3ee688-6b05-4386-b699-e081502d0d56",
    "disk.read.bytes": "c55b7c82-6739-4b27-8871-110031824730",
    "memory.swap.out": "2aec7c5-1103-4b6b-a2df-a9f905757762",
    "cpu": "8b1e749d-c055-44a8-845e-1d8f7d30cbda"
},
"original_resource_id": "8e8ca3d6-6b1c-4a97-a242-ec8549f6628e",
"revision_start": "2018-07-04T10:23:43.925047+00:00",
"project_id": "3b328fd576c84f098df8a605e71bdf59",
"type": "instance",
"id": "8e8ca3d6-6b1c-4a97-a242-ec8549f6628e"
}
},
]

```

2.2. List the details for the **disk.read.requests.rate** metric.

```
[student@workstation ~](developer1-finance)]$ openstack metric show \
--resource-id 8e8ca3d6-6b1c-4a97-a242-ec8549f6628e \
disk.read.requests.rate
+-----+-----+
| Field | Value |
+-----+-----+
| archive_policy/name | high |
| creator | ef6cde08413746d6b0a1dea1a8565522:553a5434e6 |
| | 3842eb80513c9e740b61e7 |
| id | 74678be7-239c-455d-8eba-da6ea9ea7392 |
| name | disk.read.requests.rate |
| resource/created_by_project_id | 553a5434e63842eb80513c9e740b61e7 |
| resource/created_by_user_id | ef6cde08413746d6b0a1dea1a8565522 |
| resource/creator | ef6cde08413746d6b0a1dea1a8565522:553a5434e6 |
| | 3842eb80513c9e740b61e7 |
| resource/ended_at | None |
| resource/id | 8e8ca3d6-6b1c-4a97-a242-ec8549f6628e |
| resource/original_resource_id | 8e8ca3d6-6b1c-4a97-a242-ec8549f6628e |
| resource/project_id | 3b328fd576c84f098df8a605e71bdf59 |
| resource/revision_end | None |
| resource/revision_start | 2018-07-04T05:01:08.201559+00:00 |
| resource/started_at | 2018-07-04T05:01:08.201548+00:00 |
| resource/type | instance |
| resource/user_id | e676507192d848a19f067ccdc0a20bd0 |
| unit | None |
+-----+-----+
```

The **disk.read.requests.rate** metric uses the **high** archive policy. The **high** archive policy uses as low as one second granularity for aggregation and the maximum life span of the aggregated data is 365 days.

2.3. Display the measures gathered and aggregated by the **disk.read.requests.rate** metric associated with the **finance-server1** instance. The number of records returned in the output may vary. In the output

screen, **8e8ca3d6-6b1c-4a97-a242-ec8549f6628e** is the ID of the **finance-server1** instance.



IMPORTANT

In the classroom environment, the metric service takes 15 minutes to gather and compute first set of metering samples.

```
[student@workstation ~ (developer1-finance)]$ openstack metric measures show \
--resource-id 8e8ca3d6-6b1c-4a97-a242-ec8549f6628e \
disk.read.requests.rate
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2018-07-04T10:30:00+05:30 | 3600.0 | 0.00166716985 |
| 2018-07-04T10:42:00+05:30 | 60.0 | 0.0016675937 |
| 2018-07-04T10:52:00+05:30 | 60.0 | 0.001666746 |
| 2018-07-04T10:42:50+05:30 | 1.0 | 0.0016675937 |
| 2018-07-04T10:52:50+05:30 | 1.0 | 0.001666746 |
+-----+-----+-----+
```

Observe the **value** column, which displays the aggregated values based on archive policy associated with the metric. The **3600**, **60**, and **1** in the **granularity** column values represent the aggregation periods of 1 day, 1 hour, and 1 minute, respectively.

- 2.4. Using the resource ID, list the maximum measures associated with the **cpu_util** metric with **one** minute granularity. The number of records returned in the output may vary.

```
[student@workstation ~ (developer1-finance)]$ openstack metric measures show \
--resource-id 8e8ca3d6-6b1c-4a97-a242-ec8549f6628e \
--aggregation max \
--granularity 60 \
cpu_util
+-----+-----+-----+
| timestamp | granularity | value |
+-----+-----+-----+
| 2018-07-04T10:42:00+05:30 | 60.0 | 0.2609810508 |
| 2018-07-04T10:52:00+05:30 | 60.0 | 0.2350065626 |
| 2018-07-04T11:02:00+05:30 | 60.0 | 0.248329562 |
+-----+-----+-----+
```

- 3. List the average CPU utilization for all instances provisioned and limit the query to the past 60 minutes data. Query for all instances belonging to the **finance** server group.
- 3.1. List the attributes supported by the **instance** resource type. The command returns the attributes that may be used to query this resource type.

```
[student@workstation ~ (developer1-finance)]$ openstack metric resource-type \
show instance
+-----+-----+-----+
| Field | Value |
+-----+-----+
| attributes/display_name | max_length=255, min_length=0, required=True, |
|
```

	type=string	
attributes/flavor_id	max_length=255, min_length=0, required=True,	
	type=string	
attributes/flavor_name	max_length=255, min_length=0, required=True,	
	type=string	
attributes/host	max_length=255, min_length=0, required=True,	
	type=string	
attributes/image_ref	max_length=255, min_length=0, required=False,	
	type=string	
attributes/server_group	max_length=255, min_length=0, required=False,	
	type=string	
name	instance	
state	active	
+-----+-----+-----+		

- 3.2. Only users with the admin role map query measures using resource attributes. Use the **architect1** user's Identity credentials to execute the command. The **architect1** credentials are stored in the **/home/student/architect1-finance-rc** file.

```
[student@workstation ~-(developer1-finance)]$ source ~/architect1-finance-rc
[student@workstation ~-(architect1-finance)]$
```

- 3.3. List the metadata properties associated with the **finance-server1** instance.

```
[student@workstation ~-(architect1-finance)]$ openstack server show \
-c properties finance-server1
+-----+-----+
| Field      | Value
+-----+-----+
| properties | metering.server_group='finance' |
+-----+-----+
```

- 3.4. Use the **openstack metric measures aggregation** command to list the average CPU utilization for all instances. Use the **--start** option to limit the query to the past 60 minutes data. Use the **--query** option to filter the instances.

The **instance** resource type has the **server_group** attribute. The **server_group** attribute specifies the instance metadata property. Use the **--refresh** option to force aggregation of all known measures.

The time series database service expects that time series have a certain percentage of timestamps in common. This percentage is by default set to 100%. Use the **--needed-overlap** option to set the percentage to **0**.

The number of records returned in the output may vary.

```
[student@workstation ~-(architect1-finance)]$ openstack metric measures \
aggregation \
--metric cpu_util \
--aggregation mean \
--resource-type instance \
--start $(date -u --iso=seconds -d "-60mins") \
--query "server_group='finance'" \
--needed-overlap 0 \
--refresh
+-----+-----+-----+
```

timestamp	granularity	value
2018-07-04T13:30:00+05:30	3600.0	0.255832388217
2018-07-04T14:30:00+05:30	3600.0	0.184584744158
2018-07-04T14:32:00+05:30	60.0	0.1858290391
2018-07-04T14:42:00+05:30	60.0	0.1401828896
2018-07-04T14:52:00+05:30	60.0	0.0891666539
2018-07-04T15:02:00+05:30	60.0	0.1041650796
2018-07-04T15:12:00+05:30	60.0	0.10374118845
2018-07-04T15:22:00+05:30	60.0	0.4758452247
2018-07-04T14:32:50+05:30	1.0	0.1858290391
2018-07-04T14:42:48+05:30	1.0	0.1401828896
2018-07-04T14:52:48+05:30	1.0	0.0891666539
2018-07-04T15:02:48+05:30	1.0	0.0858325314
2018-07-04T15:02:50+05:30	1.0	0.1224976278
2018-07-04T15:12:48+05:30	1.0	0.088331848
2018-07-04T15:12:50+05:30	1.0	0.1191505289
2018-07-04T15:22:48+05:30	1.0	0.8291784493
2018-07-04T15:22:50+05:30	1.0	0.1225120001

Cleanup

From **workstation**, run the **lab stacks-metrics cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab stacks-metrics cleanup
```

This concludes the guided exercise.

DEPLOYING A STACK

OBJECTIVES

After completing this section, students should be able to deploy a stack.

DESCRIBING THE ORCHESTRATION SERVICE

Orchestration automates provisioning of cloud resources, including virtual networks, storage, and servers. It gives developers and system administrators an easy way to create and manage cloud resources in an orderly and predictable fashion. The Orchestration service (heat) provides OpenStack with orchestration functionality. It is comprised of various services, including the engine service, the application programming interface (API) service, and a legacy integration with CloudFormation, the Amazon orchestration service. Orchestration facilitates service deployment by providing system administrators with a unified interface that can parse text-based instructions. Administrators write *templates* that specify the resources to spawn in the cloud, such as compute instances or volumes. The orchestration stack can be composed of a single template or multiple nested templates.

Heat templates include most of the resources administrators require when using command-line tools, such as **openstack server** for instances or **openstack volume** for volumes. The orchestration service provides an Auto Scaling feature for administrators to dynamically adjust their cloud resource usage. The orchestration service supports scaling in and out, effectively recycling unused resources. The benefits of using templates include:

- Provide access to all underlying service APIs.
- Modular and resource-oriented. Infrastructure elements are defined as objects.
- Recursively defined and reused as nested stacks. Orchestration stacks for cloud infrastructure to be defined and reused in a modular way.
- Pluggable resource implementation, for the definition of custom resources.
- Define high-availability functionality.
- Managed with a version control system. For maintaining a history of changes, rolling back to a previous version at any time, and recording who made changes to a template.

Heat Orchestration Service Components

The following service components make up the orchestration service:

openstack-heat-api

The Heat API service provides a REST API that forwards requests to the Heat engine using remote procedure calls (RPCs).

openstack-heat-engine

The service that applies templates, orchestrating the creation and launching of various OpenStack resources. It reports event status back to the Heat stack owner.

openstack-heat-api-cfn

The service used by CloudFormation-compatible templates to forward requests to the Heat engine using remote procedure calls.

mariadb database

The orchestration service uses the MariaDB database named **heat** as the data store. The database connection settings are configured in **/etc/heat/heat.conf**.

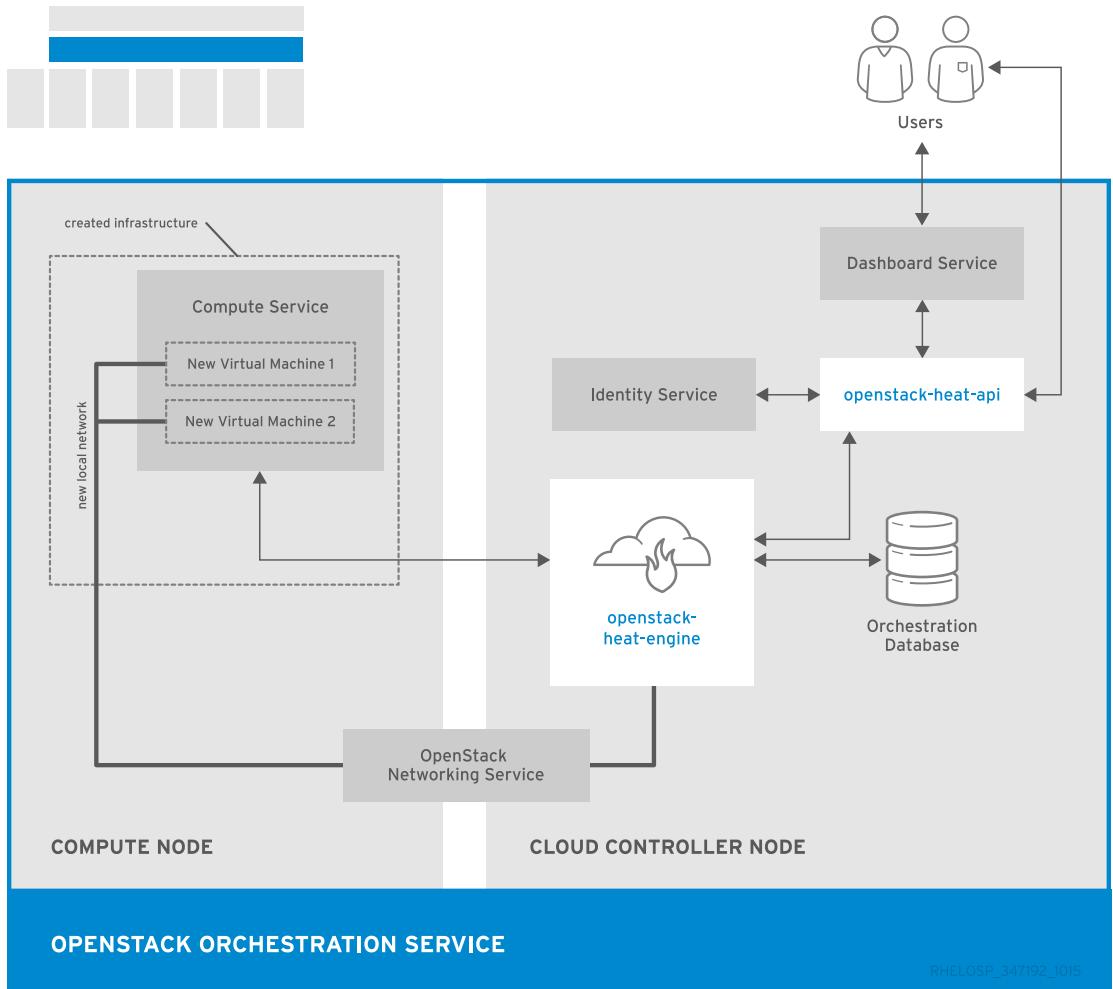


Figure 12.3: Orchestration service architecture

DEPLOYING AN ORCHESTRATION STACK

After the orchestration service is deployed, administrators can launch their orchestration *stacks*. An orchestration stack is a collection of infrastructure resources, deployed and managed through the same interface, either by using the dashboard or the command-line interface. Stacks can be used to standardize and speed up delivery, by providing a unified human-readable format. The Heat orchestration project started as an analog of AWS CloudFormation, thus making it compatible with the template formats used by CloudFormation (CFN), but it also supports its own native template format, *Heat Orchestration Template (HOT)*.

When working with the dashboard, users with the **heat_stack_owner** role can benefit from the visual features included in the dashboard for Heat. Such features include the ability to view object-based resources, using links that describe their dependencies. In the following screen, the circled objects represent Heat resources. A green object indicates the successful creation of a resource, while a red object indicates a failure. Using the dashboard provides a fast way to troubleshoot Heat resources. You can click any object to obtain more information.

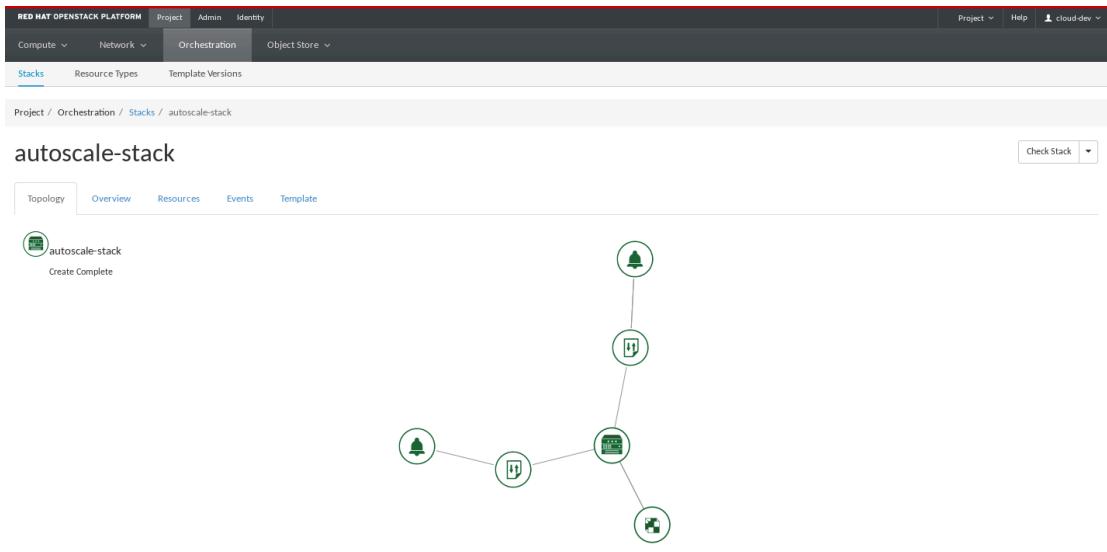


Figure 12.4: Heat stack topology in the dashboard

The orchestration service can also execute Heat Orchestration Template (HOT) files written in YAML. YAML (YAML Ain't Markup Language) format is a human-readable data serialization language. HOT is a template format designed to replace the legacy Heat CloudFormation compatible format (CFN). HOT format uses YAML syntax, and has three major sections:

Parameters

Input parameters provided when deploying from the template.

Resources

The infrastructure elements to deploy, such as virtual machines or network ports.

Outputs

Output parameters dynamically generated by Heat, available through the command-line interface, as well as the dashboard. For example, the output could provide the public IP address and name of the instance that was created from the Heat template.

ANATOMY OF A HEAT TEMPLATE

Defining the Heat Template Version

Each Heat template must include the **heat_template_version** key with the value of the current version, which is the current version of HOT. The Heat template version defines the set of functions that can be used in the templates. Available Heat template versions can be listed from the dashboard or the OpenStack CLI. To list the Heat template version using OpenStack CLI, use the **openstack orchestration template version list** command. While the description is optional, administrators should include some text that describes what users can do with the template:

```
heat_template_version: 2018-03-02
description: This is my HOT template.
```

Defining Stack Input Parameters

The **parameters** section defines the list of available parameters. For each parameter, the stack owner needs to define at least the data type, an optional default value to be used if it is not otherwise specified, an optional description, and constraints to validate the data itself:

The screen below shows the syntax for declaring a parameter in the template file:

```

parameters:
  param name:
    type: string | number | json | comma_delimited_list
    description: description of the parameter
    default: default value for parameter
    hidden: true | false
    constraints:
      parameter constraints

```

Consider the following **parameter** specification, which specifies the name of the SSH key to use, timeout, and instance metadata. The **string** data type indicates that the **key_name** parameter requires the name of the key itself. The **number** data type indicates that the **timeout** parameter requires a integer value. The **json** data type indicates that the **instance_metadata** parameter requires data defined in JSON format.

```

parameters:
  key_name:
    type: string
    description: Name of the key pair to assign to servers
  timeout:
    type: number
    description: Timeout in seconds
    default: 600
  instance_metadata:
    type: json
    description: Instance metadata
    default: { "CostCenter": { 4001 } }

```

Defining resources to be created by stack

The **resources** section of a template defines the items that will be created by Heat when a stack is deployed from the template. This may include storage, networks, ports, routers, security groups, firewall rules, or any other available resource. A resource requires a **type**, such as an instance, and various properties, all dependent on the type being used.

Use the **openstack orchestration resource type list** command to list available resource types:

```
[user@demo ~(user-demo)]$ openstack orchestration resource type list
+-----+
| Resource Type |
+-----+
...output omitted...
| OS::Neutron::Trunk |
| OS::Nova::Flavor |
| OS::Nova::FloatingIP |
| OS::Nova::FloatingIPAssociation |
| OS::Nova::HostAggregate |
| OS::Nova::KeyPair |
| OS::Nova::Quota |
| OS::Nova::Server |
| OS::Nova::ServerGroup |
| OS::Octavia::HealthMonitor |
| OS::Octavia::L7Policy |
| OS::Octavia::L7Rule |
```

```
...output omitted...
```

```
+-----+-----+
```

To show the details about the properties that must be defined to create the resource, use the **openstack orchestration resource type show** command.

```
[user@demo ~(user-demo)]$ openstack orchestration resource type show  
OS::Nova::Server
```

The screen below shows the syntax for declaring a resource in the template file:

```
resources:  
  resource ID  
    type: resource type  
    properties:  
      property name: property_value  
    # more resource specific metadata
```

Consider the following resource, which specifies the creation of an instance, using the **image**, **flavor**, and **key_name** resources to determine the image, flavor, and key pair name to use, respectively.

```
resources:  
  web_server:  
    type: OS::Nova::Server  
    properties:  
      name: Web Server  
      image: { get_param: image }  
      flavor: { get_param: flavor }  
      key_name: { get_param: key_name }  
    networks:  
      - port: { get_resource: web_server_port }
```

The **get_param** function is an intrinsic function which retrieves the value of a parameter to use during the creation of the resource. For example, if a user wants to dynamically set the password of a database, **get_param** could retrieve the password the user specified:

```
user_data:  
  str_replace:  
    template: mysql -u root -p$db_rootpassword  
    params:  
      $db_rootpassword: { get_param: db_root_password }
```

The **get_resource** function is an intrinsic function used to reference another resource within the same template. At run time, the resource is resolved to the reference ID of the referenced resource, which is resource-type specific. For example, a reference to a neutron port resource returns the respective port ID at run time.

The **str_replace** intrinsic function constructs the string dynamically based on templates with variable values provided using the **params** properties. In the above example, the **user_data** property uses the **str_replace** function to configure MySQL on a compute instance and sets the root password based on a user provided parameter. The **db_root_password** user parameter is provided by the user and is retrieved using **get_param** function. The value is stored into the

`$db_rootpassword` variable, which is used by the `str_replace` function to construct a string based on the template provided. The `$` in `$db_rootpassword` is part of the variable name.

Returning Stack Output Parameters

The `outputs` section of a template defines parameters that are available to users, either with the API or the command-line interface, after the stack has been deployed. This may include information such as IP addresses assigned to the instances, or the password that has been injected:

The screen below shows the syntax for declaring a output in the template file.

```
outputs:  
  parameter_name:  
    description: description  
    value: parameter_value
```

Consider the following example that substitutes the `host` variable with the floating IP address attribute of the `web_server_floating_ip` resource. The value for the `Login_URL` parameter is generated by the `str_replace` intrinsic function:

```
outputs:  
  login_URL:  
    description: The web server URL  
    value:  
      str_replace:  
        template: http://host  
        params:  
          host: { get_attr: [ web_server_floating_ip, floating_ip_address ] }
```

The above example uses the `get_attr` intrinsic function, which refers to the `floating_ip_address` attribute of the `web_server_floating_ip` resource. The value retrieved is stored in the `host` variable, which is used by the `str_replace` function to construct a string based on the template provided.

NOTE

The complete list of resources and their available properties can be found in the Heat Orchestration Template (HOT) specification page. The URL for this page is found in the list of references at the end of this section.

Creating Heat Environment Files

The values for parameters passed to the HOT template can be specified manually, or using an *environment file*. The environment file must be written in YAML format. Using an environment file lets you define common stack parameters so they do not need to be passed every time the stack is created. You can specify an environment file when you create or update the stack. Global environment files can also be created by the user under the default location: `/etc/heat/environment.d`. The values passed in the environment file override the existing resources. For example, users can specify their own password in the environment file and use it when deploying the stack.

In the previous Heat template example, the parameter required to execute the template was `key_name`. To specify the key name in an environment file use the following syntax:

```
parameters:
```

```
key_name: my_user_key
```

When deploying the stack, the environment file is specified as an argument to the **--environment** option to the **openstack stack create** command.

Creating Custom Resource Types

Nested stack resources enable you to specify a URL of another heat template, which is used to create another orchestration stack. The disadvantage of this approach is that the nested stack reference is based on hard-coded URLs of the template.

```
resource:  
  nested:  
    type: OS::Heat::Stack  
    properties:  
      template: "http://demo.example.com/web_server.yaml"
```

To avoid specifying the template URL inside the HOT template, the environment file can be used. The environment file is used to map resource names to templates and thus allows it to be separate from the template. The **resource_registry** section is used to map resource names to template files.

Create a template file that defines the resource, and provide the URL to the template file in the environment file to map to a custom resource type:

```
resource_registry:  
  My::Server::Custom::WebServer: "web_server.yaml"
```

The following template creates an instance, attaches a Neutron port to it, and assigns a security group to it. It also outputs the floating IP address that has been attached to the instance.

```
heat_template_version: 2018-03-02  
  
description: >  
  My HOT template.  
parameters:  
  instance_name:  
    type: string  
    default: demo-instance1  
    description: >  
      name of the instance  
  public_net:  
    type: string  
    default: provider-datacentre  
    description: >  
      name of public network for which floating IP addresses will be allocated  
  private_net:  
    type: string  
    default: demo-network1  
    description: name of private network into which servers get deployed  
  private_subnet:  
    type: string  
    default: demo-subnet1  
    description: name of private sub network into which servers get deployed
```

```

resources:
  my_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: instance_name }
      image: rhel7
      flavor: default
      key_name: example-keypair
      networks:
        - port: { get_resource: my_server_port }

  my_server_port:
    type: OS::Neutron::Port
    properties:
      network: { get_param: private_net }
      fixed_ips:
        - subnet: { get_param: private_subnet }
      security_groups: [{ get_resource: server_security_group }]

  my_server_floating_ip:
    type: OS::Neutron::FloatingIP
    properties:
      floating_network: { get_param: public_net }
      port_id: { get_resource: my_server_port }

  server_security_group:
    type: OS::Neutron::SecurityGroup
    properties:
      description: Add security group rules for server
      name: security-group
      rules:
        - remote_ip_prefix: 0.0.0.0/0
          protocol: tcp
          port_range_min: 22
          port_range_max: 22
        - remote_ip_prefix: 0.0.0.0/0
          protocol: icmp

outputs:
  my_server_public_ip:
    description: Floating IP address of My Server
    value: { get_attr: [ my_server_floating_ip, floating_ip_address ] }

```

Creating Resources using a Heat Template from Dashboard

The following steps describe the process for creating OpenStack resources using a Heat template from dashboard.

1. Log in to the dashboard as an administrator, or as a user with the **heat_stack_owner** role. In the dashboard, navigate to Project → Orchestration → Stacks.
2. Click Launch Stack.
3. Choose the Template Source. This could be a file, direct input, or a URL.
Optionally, choose the Environment Source from the list.
4. Click Next.

5. Enter the name of the stack in the **Stack Name** field. Optionally, update the **Creation Timeout (minutes)** to set a new timeout for creation of stack.

To enable the removal of resources if the stack is not correctly created, select **Rollback On Failure**.

Enter the password to be used to launch the stack in the **Password for user *username*** field.

The other fields are auto-populated with the default values from the stack template.

6. Click **Launch**.

7. Select the stack name link to view more details about the stack.

The **Topology** tab shows various resources that the stack creates. Hovering the mouse over the resource icons in the **Topology** tab shows the resource type and the resource name.

Click the **Overview** tab to display the stack parameters, stack status, stack output, and launch parameters.

The **Resources** tab shows the resources created by the stack. Click a resource name or resource ID to display more details about the resource.

The **Events** tab shows events that occur during the deployment of the stack, and after it has been deployed.

The **Template** tab shows the Heat template used to deploy the stack.

8. Navigate to Project → Orchestration → Stacks, and select the stack to be deleted. Click **Delete Stack** under the Actions column.

Managing Stacks Using the OpenStack CLI

Red Hat OpenStack Platform permits Heat stack administration from the OpenStack CLI. The **openstack stack** command allows users to create, configure, and launch stacks.

To create a stack from a template, use **openstack stack create**. The **--parameter** option is used to pass various input parameters required by the template. The option can be used multiple times to pass multiple launch parameters. The **--template** option specifies the path of the template file.

```
[user@demo ~(user-demo)]$ openstack stack create \
--parameter "instance_name=first-instance" \
--template demo-template.yaml \
demo-stack
+-----+-----+
| Field | Value |
+-----+-----+
| id | 4967ca64-85bf-4d43-9555-0b038870864d |
| stack_name | demo-stack |
| description | My HOT template. |
| |
| creation_time | 2017-03-16T10:11:05Z |
| updated_time | None |
| stack_status | CREATE_IN_PROGRESS |
| stack_status_reason | Stack CREATE started |
+-----+-----+
```

The **openstack stack create --dry-run** command validates a template file without deploying a stack. The **--enable-rollback** option enables the rollback of created resources, in case the stack fails to create all the resources defined in the template file.

Launch parameters required by the stack can be specified in an environment file. The **--environment** option specifies the path of the environment file. Multiple environment files can be specified by using the **--environment** option multiple times.

To explore the state of the stack and to list the stacks accessible to the user, use the **openstack stack list** command.

```
[user@demo ~(user-demo)]$ openstack stack list -f json
[
  {
    "Updated Time": null,
    "Stack Status": "CREATE_COMPLETE",
    "Stack Name": "demo-stack",
    "ID": "4967ca64-85bf-4d43-9555-0b038870864d",
    "Creation Time": "2017-03-16T10:11:05Z"
  }
]
```

To view information about a stack, use the **openstack stack show *STACK-NAME*** command. The stack deploys various resources. To list these resources and their status, run the **openstack stack resource list *STACK-NAME*** command.

```
[user@demo ~(user-demo)]$ openstack stack resource list demo-stack -f json
[
  {
    "resource_status": "CREATE_COMPLETE",
    "resource_name": "my_server",
    "resource_type": "OS::Nova::Server",
    "physical_resource_id": "b5c7b96e-c585-42b5-9be4-123952235e1f",
    "updated_time": "2018-06-28T08:34:31Z"
  },
  {
    "resource_status": "CREATE_COMPLETE",
    "resource_name": "my_server_port",
    "resource_type": "OS::Neutron::Port",
    "physical_resource_id": "6f07518e-2b9c-44ad-bd0c-9ffd544d7d52",
    "updated_time": "2018-06-28T08:34:31Z"
  },
  {
    "resource_status": "CREATE_COMPLETE",
    "resource_name": "wait_condition",
    "resource_type": "OS::Heat::WaitCondition",
    "physical_resource_id": "",
    "updated_time": "2018-06-28T08:34:31Z"
  },
  {
    "resource_status": "CREATE_COMPLETE",
    "resource_name": "server_security_group",
    "resource_type": "OS::Neutron::SecurityGroup",
    "physical_resource_id": "c63b4a5b-70f0-4e0f-a271-be9313f91e6a",
    "updated_time": "2018-06-28T08:34:31Z"
  }
]
```

```

    "resource_status": "CREATE_COMPLETE",
    "resource_name": "wait_handle",
    "resource_type": "OS::Heat::WaitConditionHandle",
    "physical_resource_id": "ddc40fac5caf4cd3ad20013c050f4bf2",
    "updated_time": "2018-06-28T08:34:31Z"
},
{
    "resource_status": "CREATE_COMPLETE",
    "resource_name": "my_server_floating_ip",
    "resource_type": "OS::Neutron::FloatingIP",
    "physical_resource_id": "2343f673-f717-425c-8de8-8e0dd88f3874",
    "updated_time": "2018-06-28T08:34:31Z"
}
]

```

To view details of a particular resource created by the stack, run **openstack stack resource show *STACK-NAME RESOURCE-NAME***.

```
[user@demo ~(user-demo)]$ openstack stack resource show demo-stack my_server
```

Events are generated while deploying the stack and during its life cycle. To list the events generated by the stack, use the **openstack stack event list *STACK-NAME*** command.

```
[user@demo ~(user-demo)]$ openstack stack event list demo-stack
2018-06-28 08:34:31Z [demo-stack]: CREATE_IN_PROGRESS Stack CREATE started
2018-06-28 08:34:31Z [demo-stack.server_security_group]: CREATE_IN_PROGRESS state
changed
2018-06-28 08:34:32Z [demo-stack.wait_handle]: CREATE_IN_PROGRESS state changed
2018-06-28 08:34:32Z [demo-stack.server_security_group]: CREATE_COMPLETE state
changed
2018-06-28 08:34:32Z [demo-stack.my_server_port]: CREATE_IN_PROGRESS state changed
2018-06-28 08:34:34Z [demo-stack.wait_handle]: CREATE_COMPLETE state changed
2018-06-28 08:34:34Z [demo-stack.wait_condition]: CREATE_IN_PROGRESS state
changed
2018-06-28 08:34:34Z [demo-stack.my_server_port]: CREATE_COMPLETE state changed
2018-06-28 08:34:35Z [demo-stack.my_server_floating_ip]: CREATE_IN_PROGRESS state
changed
2018-06-28 08:34:38Z [demo-stack.my_server_floating_ip]: CREATE_COMPLETE state
changed
2018-06-28 08:34:38Z [demo-stack.my_server]: CREATE_IN_PROGRESS state changed
2018-06-28 08:35:21Z [demo-stack.my_server]: CREATE_COMPLETE state changed
2018-06-28 08:35:57Z [demo-stack.wait_handle]: SIGNAL_COMPLETE Signal:
status:SUCCESS reason:Signal 1 received
2018-06-28 08:35:58Z [demo-stack.wait_condition]: CREATE_COMPLETE state changed
2018-06-28 08:35:58Z [demo-stack]: CREATE_COMPLETE Stack CREATE completed
successfully
```

The **CREATE_COMPLETE** state indicates that the resource specified in the stack is created. The **CREATE_IN_PROGRESS** state indicates that the resource creation is still in progress. The **CREATE_FAILED** state indicates that the provision of resource failed.

Detailed information about an event can be viewed using the **openstack stack event show *STACK-NAME RESOURCE-NAME EVENT-ID*** command.



NOTE

Because event IDs are not displayed by **openstack stack event list**, use the **heat event-list STACK-NAME** command to view the event ID.

List the event IDs using the **heat event-list** command.

```
[user@demo ~(user-demo)]$ heat event-list demo-stack
+-----+-----+-----+-----+
| resource_name      | id      | resource_status_reason | resource_status   |
+-----+-----+-----+-----+
| demo-stack        | e(...) | Stack CREATE started    | CREATE_IN_PROGRESS |
| server_security_group | 1(...) | state changed          | CREATE_IN_PROGRESS |
| server_security_group | 1(...) | state changed          | CREATE_COMPLETE    |
| my_server_port     | d(...) | state changed          | CREATE_IN_PROGRESS |
| my_server_port     | 6(...) | state changed          | CREATE_COMPLETE    |
| my_server_floating_ip | e(...) | state changed          | CREATE_IN_PROGRESS |
| my_server          | 3(...) | state changed          | CREATE_IN_PROGRESS |
| my_server_floating_ip | 2(...) | state changed          | CREATE_COMPLETE    |
| my_server          | 7(...) | state changed          | CREATE_COMPLETE    |
| demo-stack         | 4(...) | Stack CREATE completed | CREATE_COMPLETE    |
|                         |         | successfully            |                   |
+-----+-----+-----+-----+
```

List the detailed information of a particular event using the **openstack event show** command. Use the event ID returned by the **heat event-list** command.

```
[user@demo ~(user-demo)]$ openstack event show demo-stack
my_server 786a3e48-30e6-46d2-8259-54894ecfd3a3
+-----+
+
| Field           | Value
|
+-----+
+
...output omitted...
| resource_properties | {
|
|           |   "admin_pass": null,
|
|           |   "user_data_update_policy": "REPLACE",
|
|           |   "availability_zone": null,
|
|           |   "image": "f7286b09-ea66-4fc5-bc44-f22d33d6da88",
|
|           |   "user_data": "",
|
|           |   "diskConfig": null,
|
|           |   "flavor_update_policy": "RESIZE",
|
|           |   "flavor": "6ef2244a-d9d4-48a0-8988-c9a905065ad4",
|
```

```
|           |   "reservation_id": null,  
|           |  
|           |   "networks": [  
|           |       {  
|           |           |   "subnet": null,  
|           |           |   "uuid": null,  
|           |           |   "fixed_ip": null,  
|           |           |   "floating_ip": null,  
|           |           |   "port_extra_properties": null,  
|           |           |   "port": "d638900f-3f78-494e-858e-d25cc3a3ce0f",  
|           |           |   "network": null  
|           |       }  
|           |   ],  
|           |  
|           |   "security_groups": [],  
|           |  
|           |   "scheduler_hints": null,  
|           |  
|           |   "metadata": null,  
|           |  
|           |   "personality": {},  
|           |  
|           |   "user_data_format": "HEAT_CFNTOOLS",  
|           |  
|           |   "admin_user": null,  
|           |  
|           |   "block_device_mapping": null,  
|           |  
|           |   "key_name": "dev-keypair",  
|           |  
|           |   "software_config_transport": "POLL_SERVER_CFN",  
|           |  
|           |   "name": "demo-instance1",  
|           |  
|           |   "block_device_mapping_v2": null,  
|           |  
|           |   "image_update_policy": "REBUILD",  
|           |  
|           |   "config_drive": null  
|           |  
|           | }  
|  
| id          | 786a3e48-30e6-46d2-8259-54894ecfd3a3  
|  
| resource_type | OS::Nova::Server
```

```
+-----+  
+-----+
```

Updating a deployed stack only updates the resources that are modified in the template file. Use the **openstack stack update** command to update an existing stack.

While updating a stack, some resources are updated by changing their properties. This type of change does not require any downtime. However, there are some resources that are replaced with new resource when the stack is updated. Such an update should be avoided if possible because it requires downtime.

```
[user@demo ~(user-demo)]$ openstack stack update \  
--parameter "instance_name=demo-server1" \  
--template demo-template.yaml \  
demo-stack  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| id | 4967ca64-85bf-4d43-9555-0b038870864d |  
| stack_name | demo-stack |  
| description | My HOT template. |  
| | |  
| creation_time | 2018-06-28T08:54:31Z |  
| updated_time | 2018-06-28T08:54:31Z |  
| stack_status | UPDATE_IN_PROGRESS |  
| stack_status_reason | Stack UPDATE started |  
+-----+-----+
```

Administrators, or users with **heat_stack_owner** role, can delete a stack using **openstack stack delete STACK-NAME**. All resources associated with the stack are also deleted. The **--yes** option can be used to skip the prompt when deleting the stack.

```
[user@demo ~(user-demo)]$ openstack stack delete demo-stack  
Are you sure you want to delete this stack(s) [y/N]? y
```

Managing Heat Stack Using the OpenStack CLI

The following steps outline the process for managing, viewing, and deleting a stack using the OpenStack CLI.

1. Source the identity environment file for the administrator or the user with the **heat_stack_owner** role.
2. Store the template file and required environment file on the local disk where the credentials are sourced.
3. To launch the stack, use the **openstack stack create** command.
To view detailed information about the deployed stack, use the **openstack stack show** command.
4. To view the status of the launched stack, use the **openstack stack list** command.
5. To view the resources along with their status, use the **openstack stack resource list** command.

To view the events generated by the stack, use the **openstack stack event list** command.

6. To update the stack, use the **openstack stack update** command. Optionally, verify that the update is complete using the **openstack stack list** command.

Optionally, view resource details using **openstack stack resource show** to confirm the resource was updated.
7. If you need to delete the stack, use the **openstack stack delete** command.



REFERENCES

Further information is available in the OpenStack Resource Type section of the *Template Guide* at
https://docs.openstack.org/developer/heat/template_guide

DEPLOYING A STACK

In this exercise, you will deploy a stack using a Heat orchestration template to launch a web server instance. Use the dashboard to view the stack information.

OUTCOMES

You should be able to:

- Inspect the template to determine required parameters.
- Create an environment file with required parameters.
- Deploy an orchestration stack using a template.
- List resources deployed by the stack.
- View stack information from the dashboard.

Ensure that the **workstation** and overcloud's virtual machines are started.

On **workstation**, run the **lab stacks-deploy setup** command, which ensures that the environment is properly configured for this exercise.

```
[student@workstation ~]$ lab stacks-deploy setup
```

- ▶ 1. On **workstation**, download the Heat template from http://materials.example.com/heat/web_server_withfloatingip.yaml. Save the template under **/home/student/**.

```
[student@workstation ~]$ wget \
http://materials.example.com/heat/web_server_withfloatingip.yaml
...output omitted...
2018-07-03 09:32:01 (713 MB/s) - 'web_server_withfloatingip.yaml' saved
[3972/3972]
```

- ▶ 2. Open the file and inspect the parameters required by the **web_server_withfloatingip.yaml** template. The following output highlights the relevant resources that require a value in the environment file.

```
heat_template_version: queens
description: launching a custom web server

parameters:
  image_name:
    type: string
    default: web-image
    description: Image used for servers
  instance_name:
```

```

type: string
default: web-server
description: Name for the web server
key_name:
  type: string
  default: web-keypair
  description: SSH key to connect to the servers
instance_flavor:
  type: string
  default: m2.small
  description: flavor used by the servers
public_net:
  type: string
  default: public
  description: Name of public network into which servers get deployed
private_net:
  type: string
  default: private
  description: Name of private network into which servers get deployed
private_subnet:
  type: string
  default: private_subnet
  description: Name of private subnet into which servers get deployed
metadata:
  type: json
  default: { }
instance_metadata:
  type: json
  default: { }

...output omitted...

```

- 3. Source the `/home/student/developer1-finance-rc` identity environment file for the **developer1** user.

Use the **openstack** command to retrieve information about the environment. You will use this information in the next step to populate the `environment.yaml` Heat environment file to be used by the `web_server_withfloatingip.yaml` template.

3.1. Source the **developer1** user identity environment file:

```
[student@workstation ~]$ source ~/developer1-finance-rc
```

3.2. Retrieve the name of the image to be used for provisioning the instance:

```
[student@workstation ~(developer1-finance)]$ openstack image list \
-c Name
+-----+
| Name |
+-----+
| rhel7 |
| ...   |
```

```
+-----+
```

- 3.3. Retrieve the name of the key pair to be associated with the instance:

```
[student@workstation ~(developer1-finance)]$ openstack keypair list \
-c Name
+-----+
| Name      |
+-----+
| example-keypair |
+-----+
```

- 3.4. Retrieve the list of flavors (you will use the **default** flavor):

```
[student@workstation ~(developer1-finance)]$ openstack flavor list \
-c Name
+-----+
| Name      |
+-----+
| default   |
+-----+
```

- 3.5. Retrieve the name of the external network:

```
[student@workstation ~(developer1-finance)]$ openstack network list \
-c Name --external
+-----+
| Name      |
+-----+
| provider-datacentre |
+-----+
```

- 3.6. Retrieve the name of the project network:

```
[student@workstation ~(developer1-finance)]$ openstack network list \
-c Name
+-----+
| Name      |
+-----+
| provider-datacentre |
| finance-network1 |
+-----+
```

- 3.7. Retrieve the name of the subnet for the project network:

```
[student@workstation ~(developer1-finance)]$ openstack subnet list \
-c Name
+-----+
| Name      |
+-----+
| finance-subnet1 |
| provider-subnet-172.25.250 |
+-----+
```

- 4. Create the environment file for the stack, called **/home/student/environment.yaml**. Include the following resources in the environment file:

Parameters for launching the stack

PARAMETER	VALUE
<i>image_name</i>	rhel7
<i>instance_name</i>	finance-server1
<i>key_name</i>	example-keypair
<i>instance_flavor</i>	default
<i>public_net</i>	provider-datacentre
<i>private_net</i>	finance-network1
<i>private_subnet</i>	finance-subnet1
<i>instance_metadata</i>	{ "project.name": { "finance" } }

- 4.1. Create the **/home/student/environment.yaml** environment file and add the following parameters:

```
parameters:  
  image_name: "rhel7"  
  instance_name: "finance-server1"  
  key_name: "example-keypair"  
  instance_flavor: "default"  
  public_net: "provider-datacentre"  
  private_net: "finance-network1"  
  private_subnet: "finance-subnet1"  
  instance_metadata: { "project.name": { "finance" } }
```

Confirm that the environment file is properly indented.

- 5. Using the **environment.yaml** environment file and the **web_server_withfloatingip.yaml** template, execute a dry run to test creating the stack. Name the stack **webstack**.

Analyze the output of the dry run; examine the resources and attributes that will be created when you launch the stack. View the **outputs** field in the output of the dry run to determine the output parameters of the stack.

```
[student@workstation ~-(developer1-finance)]$ openstack stack create \  
--environment ~/environment.yaml \  
--template ~/web_server_withfloatingip.yaml \  
--dry-run \  
webstack  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| outputs | [ |  
| | { |
```

```
|     "output_value": null,  
|     "output_key": "web_private_ip",  
|     "description": "IP address of first web server  
|                     in private network"  
|   },  
|   {  
|     "output_value": null,  
|     "output_key": "web_public_ip",  
|     "description": "Floating IP address of the web  
|                     server"  
|   },  
|   {  
|     "output_value": "http:///",  
|     "output_key": "website_url",  
|     "description": "This URL is the \"external\" URL  
|                     that can be used to  
|                     access the web server.\n"  
|   }  
| ]  
...output omitted...  
resources [  
  {  
    "resource_name": "web_server",  
    "resource_identity": {  
      "stack_name": "webstack",  
      "stack_id": "None",  
      "project": "2b4e9e9cc9854c28b34fa9ee40a636fb",  
      "path": "/resources/web_server"  
    },  
...output omitted...  
    "properties": {  
      "rules": [  
        {  
          "remote_group_id": null,  
          "direction": "ingress",  
          "protocol": "tcp",  
          "ethertype": "IPv4",  
          "port_range_max": 22,  
          "remote_mode": "remote_ip_prefix",  
          "port_range_min": 22,  
          "remote_ip_prefix": "0.0.0.0/0"  
        },  
        {  
          "remote_group_id": null,  
          "direction": "ingress",  
          "protocol": "tcp",  
          "ethertype": "IPv4",  
          "port_range_max": 80,  
          "remote_mode": "remote_ip_prefix",  
          "port_range_min": 80,  
          "remote_ip_prefix": "0.0.0.0/0"  
        },  
        {  
          "remote_group_id": null,  
          "direction": "ingress",  
          "protocol": "icmp",  
        }  
      ]  
    }  
  }  
]
```

```

|           |           "ethertype": "IPv4",
|           |           "port_range_max": null,
|           |           "remote_mode": "remote_ip_prefix",
|           |           "port_range_min": null,
|           |           "remote_ip_prefix": "0.0.0.0/0"
|           |
|           |
|           ],
|           "name": "dev-secgroup",
|           "description": "Add security group rules for
|                           the multi-tier ar...re"
|           },
|           "resource_type": "OS::Neutron::SecurityGroup",
|           "metadata": {}
|       },
...output omitted...

```

- ▶ 6. Launch the stack using the `/home/student/web_server_withfloatingip.yaml` template and the `/home/student/environment.yaml` environment file. Call the stack **webstack**.

Use the `--enable-rollback` option to roll back the resources created in case of failure.

```

[student@workstation ~ (developer1-finance)]$ openstack stack create \
--environment ~/environment.yaml \
--template ~/web_server_withfloatingip.yaml \
--enable-rollback \
--wait \
webstack
...output omitted...
+-----+-----+
| Field      | Value   |
+-----+-----+
| id          | 85407ca4-893a-4e83-b902-59373f58d7f7 |
| stack_name  | webstack |
| description  | launching a custom web server |
| creation_time | 2018-06-28T08:34:30Z |
| updated_time | None    |
| stack_status | CREATE_COMPLETE |
| stack_status_reason | Stack CREATE completed successfully |
+-----+-----+

```

- ▶ 7. List the resources created by the **webstack** stack.

7.1. List the resources created by the stack:

```

[student@workstation ~ (developer1-finance)]$ openstack stack resource list \
webstack -f json
[
  {
    "resource_status": "CREATE_COMPLETE",
    "resource_name": "web_server",
    "resource_type": "OS::Nova::Server",
    "physical_resource_id": "f14e45ce-23eb-4f4a-a14b-1ffca61ed0ca",
    "updated_time": "2018-06-28T08:34:31Z"
  },

```

```

    },
    {
        "resource_status": "CREATE_COMPLETE",
        "resource_name": "web_net_port",
        "resource_type": "OS::Neutron::Port",
        "physical_resource_id": "366fdd55-27d9-4d87-bbc6-9a986a1d2078",
        "updated_time": "2018-06-28T08:34:31Z"
    },
    {
        "resource_status": "CREATE_COMPLETE",
        "resource_name": "wait_condition",
        "resource_type": "OS::Heat::WaitCondition",
        "physical_resource_id": "",
        "updated_time": "2018-06-28T08:34:31Z"
    },
    {
        "resource_status": "CREATE_COMPLETE",
        "resource_name": "web_security_group",
        "resource_type": "OS::Neutron::SecurityGroup",
        "physical_resource_id": "c2d57d54-6f13-4d19-944c-135607121678",
        "updated_time": "2018-06-28T08:34:31Z"
    },
    {
        "resource_status": "CREATE_COMPLETE",
        "resource_name": "wait_handle",
        "resource_type": "OS::Heat::WaitConditionHandle",
        "physical_resource_id": "ddc40fac5caf4cd3ad20013c050f4bf2",
        "updated_time": "2018-06-28T08:34:31Z"
    },
    {
        "resource_status": "CREATE_COMPLETE",
        "resource_name": "web_floating_ip",
        "resource_type": "OS::Neutron::FloatingIP",
        "physical_resource_id": "f996d2bb-9cf9-4877-ae90-ca0a74f0bf41",
        "updated_time": "2018-06-28T08:34:31Z"
    }
]

```

7.2. List the events generated by the stack.

```
[student@workstation ~ (developer1-finance)]$ openstack stack event list \
webstack
2018-06-28 08:34:31Z [webstack]: CREATE_IN_PROGRESS Stack CREATE started
2018-06-28 08:34:31Z [webstack.web_security_group]: CREATE_IN_PROGRESS state
changed
2018-06-28 08:34:32Z [webstack.wait_handle]: CREATE_IN_PROGRESS state changed
2018-06-28 08:34:32Z [webstack.web_security_group]: CREATE_COMPLETE state changed
2018-06-28 08:34:32Z [webstack.web_net_port]: CREATE_IN_PROGRESS state changed
2018-06-28 08:34:34Z [webstack.wait_handle]: CREATE_COMPLETE state changed
2018-06-28 08:34:34Z [webstack.wait_condition]: CREATE_IN_PROGRESS state changed
2018-06-28 08:34:34Z [webstack.web_net_port]: CREATE_COMPLETE state changed
2018-06-28 08:34:35Z [webstack.web_floating_ip]: CREATE_IN_PROGRESS state changed
2018-06-28 08:34:38Z [webstack.web_floating_ip]: CREATE_COMPLETE state changed
2018-06-28 08:34:38Z [webstack.web_server]: CREATE_IN_PROGRESS state changed
2018-06-28 08:35:21Z [webstack.web_server]: CREATE_COMPLETE state changed
2018-06-28 08:35:57Z [webstack.wait_handle]: SIGNAL_COMPLETE Signal:
status:SUCCESS reason:Signal 1 received
```

```
2018-06-28 08:35:58Z [webstack.wait_condition]: CREATE_COMPLETE state changed
2018-06-28 08:35:58Z [webstack]: CREATE_COMPLETE Stack CREATE completed
successfully
```

- 8. Verify that the **web_server** resource was created by the **webstack** stack. Use **curl** to view the web page served by the instance.
- 8.1. List the instances present in the environment to check the status of the **finance-server1** instance:

```
[student@workstation ~ (developer1-finance)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| finance-server1 | ACTIVE |
+-----+-----+
```

- 8.2. Wait until the **finance-server1** instance status changes to **ACTIVE**. View the complete details for the **web_server** resource created by the stack. Note the floating IP address allocated to the instance and the instance name.

```
[student@workstation ~ (developer1-finance)]$ openstack stack resource show \
-f json webstack web_server
{
    "resource_name": "web_server",
    "description": "",
    ...output omitted...
    "attributes": {
        "OS-EXT-STS:task_state": null,
        "addresses": {
            "finance-network1": [
                {
                    "OS-EXT-IPS-MAC:mac_addr": "fa:16:3e:45:67:84",
                    "version": 4,
                    "addr": "192.168.1.X",
                    "OS-EXT-IPS:type": "fixed"
                },
                {
                    "OS-EXT-IPS-MAC:mac_addr": "fa:16:3e:45:67:84",
                    "version": 4,
                    "addr": "172.25.250.P",
                    "OS-EXT-IPS:type": "floating"
                }
            ],
            ...
        },
        ...output omitted...
        "status": "ACTIVE",
        "updated": "2018-06-28T08:35:20Z",
        "hostId": "06910493c3dd091345b9cfa5273d2e4e4f8be9f40185119c8911ac28",
        "OS-SRV-USG:terminated_at": null,
        "key_name": "example-keypair",
        "os_collect_config": {},
        "name": "finance-server1",
        "created": "2018-06-28T08:34:40Z",
        "tenant_id": "91d634a8c1534bc1a105994bf434e53b",
        ...
    }
}
```

```

"os-extended-volumes:volumes_attached": [],
"metadata": {
    "project.name": "{\"finance\": null}"
},
"resource_type": "OS::Nova::Server"
}

```

8.3. View the output parameters and their values for the **webstack** stack:

```
[student@workstation ~-(developer1-finance)]$ openstack stack output list \
webstack
+-----+-----+
| output_key | description |
+-----+-----+
| web_private_ip | IP address of first web server in private network |
| web_public_ip | Floating IP address of the web server |
| website_url | This URL is the "external" URL |
| | can be used to access the web server. |
+-----+-----+
```

View the value for the *website_url* output parameter:

```
[student@workstation ~-(developer1-finance)]$ openstack stack output show \
webstack website_url
+-----+-----+
| Field | Value |
+-----+-----+
| description | This URL is the "external" URL |
| | that can be used to access the web server. |
| | |
| output_key | website_url |
| output_value | http://172.25.250.P/ |
+-----+-----+
```

8.4. View the web page served by the **finance-server1** instance using the output value returned by the *website_url* parameter:

```
[student@workstation ~-(developer1-finance)]$ curl http://172.25.250.P
<h1>You are connected to 172.25.250.P</h1>
<h2>The private IP address is:192.168.1.N</h2>
Red Hat Training
```

- ▶ 9. Log in to the dashboard at `http://dashboard.overcloud.example.com` using **developer1** as the user and **redhat** as the password. View the **webstack** stack information from the dashboard.
 - 9.1. Navigate to Project → Orchestration → Stacks.
 - 9.2. Click the **webstack** link to view information about the stack.
 - 9.3. Click the Topology, Overview, Resources, Events, and Template tabs to view information of the stack.
- ▶ 10. Log out of the dashboard.

Cleanup

On **workstation**, run the **lab stacks-deploy cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab stacks-deploy cleanup
```

This concludes the guided exercise.

CONFIGURING STACK AUTO SCALING

OBJECTIVES

After completing this section, students should be able to configure stack auto-scaling.

BENEFITS OF AUTO SCALING

Auto Scaling provides a means of automatically scaling a stack deployment out or in to meet service requirements. In Red Hat OpenStack Platform the orchestration service implements Auto Scaling. Consider the example of a web application whose usage at the beginning and end of the week is minimal. During the middle of the week, more users starts logging and the demand for the web application resources increases significantly. Traditionally, there are two ways to plan for the resources required.

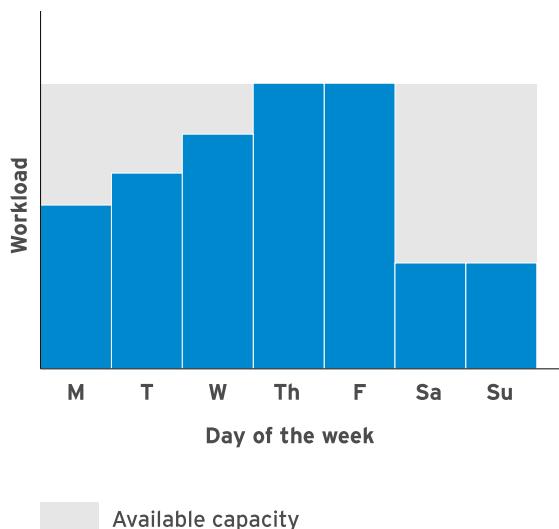


Figure 12.5: Workload pattern

One of the ways is to provision resources with extra capacity for the entire week to meet the raise in demand. The downside to this approach is that it leads to unused capacity, raising the cost of running the application.

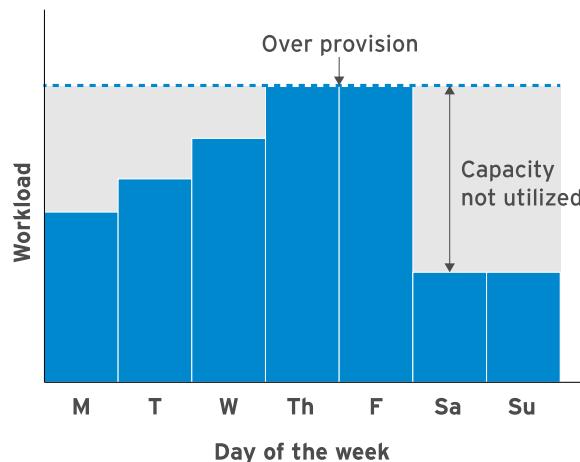


Figure 12.6: Workload pattern with extra capacity provisioned

The second option is to provision resources to handle the average demand of the application. This approach is more cost optimized, but has the risk of poor user experience when demand surges during the middle of the week.

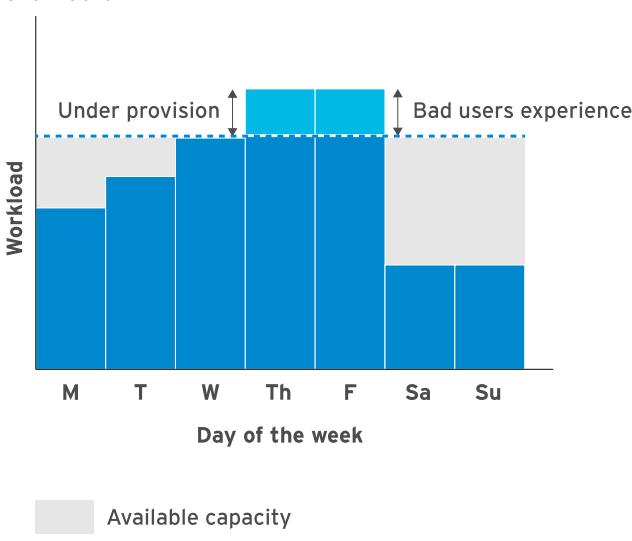


Figure 12.7: Workload pattern with average capacity provisioned

By using automatic scaling, resources are added to the web application stack only when necessary to increase the capacity to handle a surge in demand. To save cost, these resources are terminated when they are no longer required.

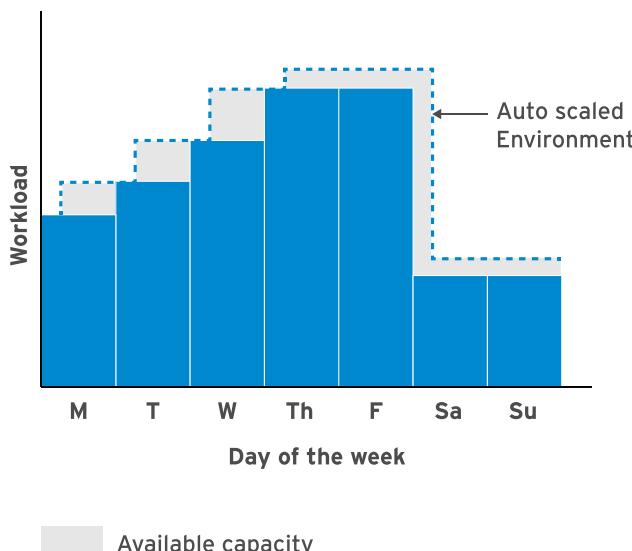


Figure 12.8: Workload pattern with auto-scaled environment

Adding automatic scaling to the cloud infrastructure allows the cloud provider to gain the following benefits:

- Auto Scaling allows cloud resources to be running with the capacity required to handle the demand.
- Auto Scaling allows a reduction of the infrastructure cost, as it can dynamically provision or terminate the resources based on demand.

Understanding How Auto Scaling Works

The Auto Scaling group defines the resource to be provisioned. It launches a number of instances defined by the desired capacity or minimum group size parameters.

Telemetry alarms are defined to trigger the auto scaling to either scale out or scale in the stack based on the alarm rules. Primarily, there are two alarms created: one for scaling out and the other for scaling in. The alarm action for these alarms invokes the URL associated with the scaling-out policy and scaling-in policy.

The Auto Scaling policy defines the number of resources that need to be added or removed in the event of scale out or scale in. It uses the defined Auto Scaling Group. Multiple Auto Scaling policies can be defined for automatically scaling the infrastructure to adjust to various usage patterns.

Consider a deployment where Heat orchestration resources are used to manage an infrastructure comprised of a public IP address, a load balancer pool, a load balancer, a set of instances, and alarming service alarms to monitor the infrastructure. The user accessing the stack generates HTTP requests using the public IP address attached to the load balancer. The load balancer processes the HTTP request and sends the request to be served by one of the web servers in the load balancer pool. The load balancer sends the metric data, such as number of HTTP requests per second, to the metering service. The alarm associated with the load balancer checks for the average HTTP requests per second.

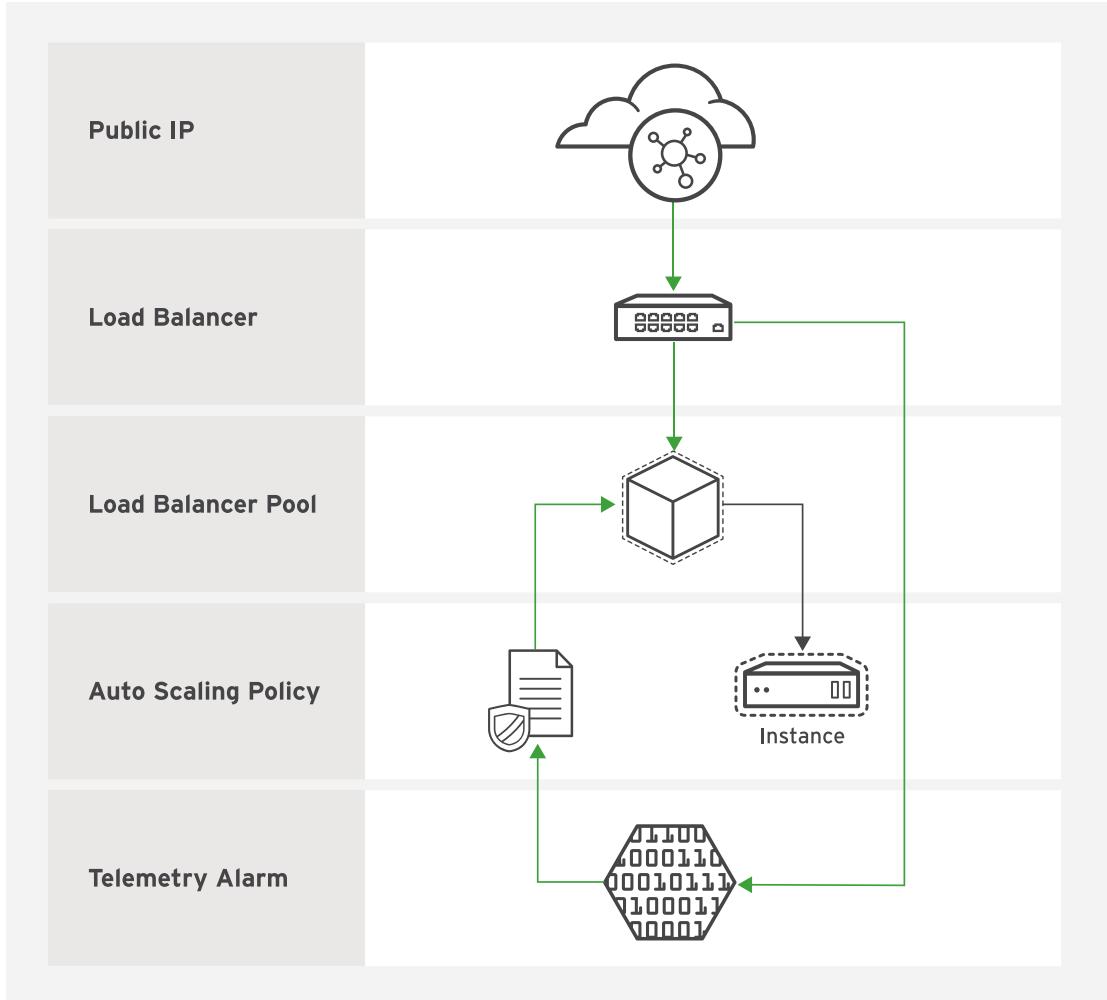


Figure 12.9: Deploying infrastructure with orchestration

As the alarm associated with the load balancer exceeds or falls below the set threshold value, a scaling event occurs. The scaling event either scales out and adds more web servers to the load balancer pool, or it scales in by terminating a web server from the pool.

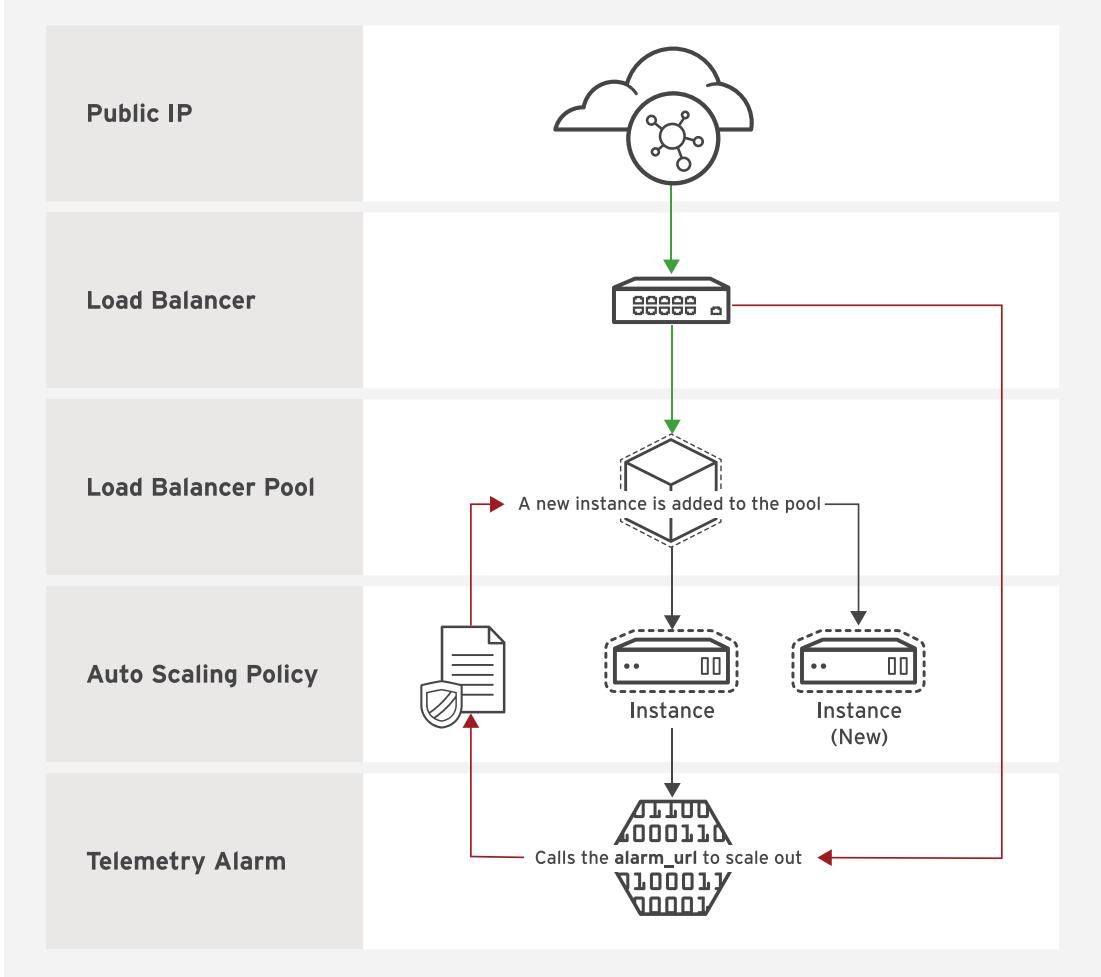


Figure 12.10: Scaling infrastructure load with orchestration

Using Auto Scaling with Orchestration Service

The orchestration service allows administrators to dynamically scale out and scale in their stacks. The orchestration service communicates with service alarms to determine the conditions under which a scaling event is triggered. Administrators can monitor various resources, including:

- CPU load
- CPU usage as a percentage
- Disk usage
- Network usage

Almost all metrics monitored by the metric service can be used to dynamically scale orchestration stacks. The following Heat orchestration resource types are used to create resources for auto-scaling:

OS::Heat::AutoScalingGroup

This resource type is used to define the auto-scaling resource group. Required properties include `max_size`, `min_size`, and `resource`. Optional properties include `cooldown`, `desired_capacity`, and `rolling_updates`.

The `resource` property defines the resource and its properties that would be created in the Auto Scaling group.

The *max_size* property defines the maximum number of identical resources in the Auto Scaling group. The *min_size* property defines the minimum number of identical resources that must be running in the Auto Scaling group.

The *desired_capacity* property defines the desired initial number of resources. If not specified, the value of *desired_capacity* is equal to the value of *min_size*.

The optional *cooldown* property defines the time gap, in seconds, between two consecutive scaling events.

The *rolling_update* property defines the sequence for rolling out the updates. It streamlines the update rather than taking down the entire service at the same time. The optional *max_batch_size* and *min_in_service* parameters of the property define maximum and minimum numbers of resources to be replaced at once. The *pause_time* property defines a time to wait between two consecutive updates.

OS::Heat::ScalingPolicy

The resource type used to define the Auto Scaling policy used to manage the scaling for the Auto Scaling group defined by *OS::Heat::AutoScalingGroup*. Required properties include *adjustment_type*, *auto_scaling_group_id*, and *scaling_adjustment*. Optional properties include *cooldown* and *min_adjustment_step*.

The Auto Scaling policy uses the *adjustment_type* property to decide on the type of adjustment needed. The value for this property can be set to *change_in_capacity*, *exact_capacity*, or *percentage_change_in_capacity*. These values set the adjustment type as either absolute or percentage.

The Auto Scaling policy uses the *auto_scaling_group_id* property to apply the policy to the Auto Scaling group. The *scaling_adjustment* property defines the size of adjustment, a positive value indicates the resources to be added. A negative value terminates the resource. The *cooldown* property defines the time gap, in seconds, between two consecutive scaling events.

The *min_adjustment_step* property is used in conjunction with the *percentage_change_in_capacity* property. The property defines the minimum number of resources added or terminated when the Auto Scaling group scales out or scales in.

OS::Aodh::GnocchiAggregationByResourcesAlarm

This resource type defines the Telemetry alarm based on the aggregation of resources. The alarm monitors the usage of all the sub-resources of a resource. Required properties include *metric*, *query*, *resource_type*, and *threshold*. Optional properties include *aggregation_method*, *alarm_actions*, *comparison_operator*, *evaluation_periods*, and *granularity*.

The *alarm_actions* property defines an action to be taken when the alarm is triggered. When the alarm associated with a scaling policy is triggered, the *alarm_actions* property calls the *signal_url* attribute of the Auto Scaling policy. The *signal_url* attribute is the URL that handles an alarm.

The *metric* property defines the metric to be monitored. The *evaluation_periods* property sets the number of periods to evaluate the metric measures before setting off the alarm. The *threshold* property defines the value which, when exceeded or reduced, the alarm is triggered.

OS::Octavia::HealthMonitor

This resource allows administrators to create and manage Octavia health monitors. A health monitor is an LBaaS service component that checks if instances are still running on a specified protocol and port. Required properties include *delay*, *max_retries*, *pool*, *timeout*, and *type*. Optional properties include *expected_parameters*, *http_method*, and *url_path*.

OS::Octavia::Pool

This resource manages Octavia LBaaS pools, which represent a group of nodes.

Pools define the subnet where nodes reside, the balancing algorithm, and the nodes themselves. Administrators can use this resource to create front-facing load balancers, themselves rerouting incoming requests to back-end servers. Required properties include *lb_algorithm*, and *protocol*. Optional properties include *session_persistence*.

OS::Octavia::LoadBalancer

This resource can be used in conjunction with the **OS::Octavia::Pool** resource to manage Octavia load balancers. It defines load balancers that route incoming requests to instances defined in a Neutron pool. Required properties include *pool_id* and *protocol_port*.

Administrators can use these resources in templates to perform various tasks. For example:

- Create a farm of web servers that automatically scales when network traffic reaches a certain threshold.
- Create a database cluster that automatically scales when the number of incoming connections reaches a certain threshold.
- Create a set of application servers that automatically scales when CPU usage exceeds a certain level.

Heat orchestration resources can dynamically retrieve values of alarms and run actions in real time, adding or removing a resource. Auto Scaling can be used to build complex environments that automatically adjust their capacity by dynamically adding or removing resources.

The following example defines an alarm resource named **disk_alarm_high**. The alarm is evaluated against the average of the **disk.read.requests.rate** meter. The meter collects the rate of change at which read requests are served by a disk connected to an instance. The alarm statistics are calculated for the data samples accumulated by the metric for a period of **60** seconds. The **disk_alarm_high** alarm is triggered when the threshold value exceeds **6000** twice, as the **evaluation_periods** is set to **2**. When the alarm is triggered, it calls the *single_url* attribute of the **scaleup_policy** scaling policy as specified in the *alarm_action* property.

```
memory_alarm_high:
  type: OS::Aodh::GnocchiAggregationByResourcesAlarm
  properties:
    description: Scale up if memory usage is 50% for 2 minutes
    metric: memory
    aggregation_method: mean
    granularity: 60
    evaluation_periods: 2
    threshold: 6000
    resource_type: instance
    comparison_operator: gt
    alarm_actions:
      - str_replace:
          template: trust+url
          params:
            url: {get_attr: [scaleup_policy, signal_url]}
    query:
      str_replace:
        template: '{"=": {"server_group": "stack_id"}}'
        params:
          stack_id: {get_param: "OS::stack_id"}
```

In the following example, the **scaleup_policy** and **scaledown_policy** defines the policy used to manage the scaling of resources. The associated Auto Scaling policy uses the **adjustment_type** property to decide on the type of the adjustment needed. The stack administrator can choose the **adjustment_type** property value as:

- **change_in_capacity**: this value is selected when the resources to be added or terminated is in absolute numbers.
- **exact_capacity**: this value is selected when existing resources are replaced with the exact number as defined by size of adjustment.
- **percentage_change_in_capacity**: this value is selected when the size of adjustment is in percentage.

The Auto Scaling policy uses the **auto_scaling_group_id** property to set to the Auto Scaling group ID of the **web_scaler** Auto Scaling group resource. The **scaling_adjustment** property in the scale up policy is set to **1** to add one resource to the Auto Scaling group. In the **scaledown_policy** Auto Scaling policy the **scaling_adjustment** is set to **-1** to terminate one resource from the Auto Scaling group. The cool down period is set to **180** seconds between two consecutive scaling events. While setting the cool down period, factors, such as the time required by an instance to serve the request from the load balancer, must be kept in mind.

```
scaleup_policy:  
  type: OS::Heat::ScalingPolicy  
  properties:  
    adjustment_type: change_in_capacity  
    auto_scaling_group_id: { get_resource: web_scaler }  
    cooldown: 180  
    scaling_adjustment: 1  
  
scaledown_policy:  
  type: OS::Heat::ScalingPolicy  
  properties:  
    adjustment_type: change_in_capacity  
    auto_scaling_group_id: { get_resource: web_scaler }  
    cooldown: 180  
    scaling_adjustment: -1
```

The **OS::Heat::AutoScalingGroup** resource type defines the Heat orchestration resource to allow creation of identical resources based on the **min_size** property. In the following example, the **resource** property defines the **My::Server::Custom::Webserver** custom resource and its attributes. The **desired_capacity** property defines the desired initial number of resources. This property is used in cases where the desired capacity is more than the minimum size, for example in an environment where demand can be anticipated.

The **max_size** property is set to **4**, so when a scale out event occurs, the number of resources can not exceed more than four. The **min_size** property is set to **1**, starting the initial Auto Scaling group with one resource. The **cooldown** property value defined in the scaling policy overrides the property value set in the Auto Scaling group.

```
web_scaler:  
  type: OS::Heat::AutoScalingGroup  
  properties:  
    desired_capacity: 2  
    cooldown: 300  
    max_size: 4
```

```
min_size: 1
resource:
    type: My::Server::Custom::WebServer
properties:
    instance_name: { demo-server }
    instance_flavor: { default }
    image_name: { rhel7-web }
    key_name: { example-keypair }
    public_net: { provider-datacentre }
    private_net: { demo-network1 }
    private_subnet: { demo-subnet1 }
    pool_name: { demo-webpool }
```

Troubleshooting Auto Scaling

When a stack with automatic scaling is deployed, useful information is logged into the log file of the orchestration service. The following log files for each orchestration service is stored in the **/var/log/heat** directory on the host where the Heat orchestration components are deployed.

heat-api.log

The log recording API calls to the orchestration service.

heat-engine.log

This log stores the processing of the Heat templates and their requests to the underlying API for the resources defined in the template.

heat-manage.log

The log of service events that occur when deploying a stack, or when a scaling event triggers.

When the Auto Scaling stack fails to deploy, use some of the **openstack stack** commands to identify the failed component. Use **openstack stack list** with **--show-nested** to view all the nested stacks. This command returns nested stack IDs, names, and stack status.

The stack ID or name of the failed nested stack can then be used to identify the failed resource. Use the **openstack stack resource list STACK-NAME** command. Resource details displayed by the command are the resource name, physical resource ID, resource type, and its status. The physical resource ID can be queried using the **openstack stack resource show** command to check the output value returned while creating the resource.

The **openstack stack event list** command returns the **SIGNAL_COMPLETE** status when a scaling event successfully completes. More information about the scaling event can be viewed using the **openstack stack event show** command.

To find the root cause for a failure, the specific log files belonging to the services whose resources are created can be viewed. For example, the log related to instances can be found under the **/var/log/nova/** directory. When the error **NoValidHost** is logged in the **nova-scheduler.log** file, it indicates that no valid host was found to launch the instance. To resolve this error, consider choosing a smaller instance size, or increase the over-committed values in the **/etc/nova/nova.conf** file. The **/var/log/cloud-init.log** log file present in the created instance can be viewed to verify the user data was successfully executed.



REFERENCES

Further information is available in the Configure auto scaling for compute section of the *Auto Scaling for Compute* for Red Hat OpenStack Platform at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html/auto_scaling_for_instances/index

CONFIGURING STACK AUTO SCALING

In this exercise, you will auto-scale a web server using the CPU utilization metrics of the web instance.

OUTCOMES

You should be able to auto-scale a pool of web servers using the telemetry metrics.

Ensure that the **workstation** and overcloud's virtual machines are started.

From **workstation**, run the **lab stacks-autoscale setup** command, which ensures that the OpenStack services are running and that the resources from the previous labs are available. The script launches the **autoscale-stack** stack, using the **environment.yaml** environment file and **webserver-autoscale.yaml** template file. The script also changes the configuration to collect metrics every 15 seconds.

```
[student@workstation ~]$ lab stacks-autoscale setup
```

- ▶ 1. On **workstation**, inspect the **webserver-autoscale.yaml** file downloaded by the setup script. Review the parameters the template file requires and the resources the stack uses. Examine the **OS::Heat::AutoScalingGroup** section that defines the rules for scaling the stack out and in.
 - 1.1. The **web_scaler** resource of the **OS::Heat::AutoScalingGroup** type is used to define the resource that is created on scale out. The properties of the **web_scaler** resource are as follows:
 - *desired_capacity*: the number of instances that should be running in the Auto Scaling group.
 - *cooldown*: the amount of time, in seconds, after a scaling activity completes before another scaling activity can start.
 - *max_size*: the maximum size of the Auto Scaling group.
 - *min_size*: the minimum size of the Auto Scaling group.

```
resources:
...output omitted...
web_scaler:
  type: OS::Heat::AutoScalingGroup
  properties:
    desired_capacity: { get_param: initial_capacity }
    cooldown: { get_param: cooldown_policy_seconds }
    max_size: { get_param: asg_group_max_size }
    min_size: { get_param: asg_group_min_size }
```

- 1.2. The **web_scaler** Auto Scaling group defines the resource that will be created on scaling. The **My::Server::Custom::WebServer** custom resource type defines

the HOT template to be used. The resource type is invoked with the properties specified to launch the instance.

```
resource:
  type: My::Server::Custom::WebServer
  properties:
    instance_name: { list_join: [ '-', [ {get_param: instance_name},
{get_attr: [random, value]} ] ] }
    instance_flavor: { get_param: instance_flavor }
    image_name: { get_param: image_name }
    key_name: { get_param: key_name }
    public_net: { get_param: public_net }
    private_net: { get_param: private_net }
    private_subnet: { get_param: private_subnet }
    instance_metadata: { "metering.server_group": {get_param:
"os::stack_id"} }
```

- 1.3. The **scaleup_policy** resource, of **OS::Heat::ScalingPolicy** type, defines the scaling out policy. The **scaledown_policy** resource, of **OS::Heat::ScalingPolicy** type, defines the scaling in policy.

The following properties define the policy:

- *adjustment_type*: defines the adjustment types.
- *auto_scaling_group_id*: the Auto Scaling group id.
- *cooldown*: overrides the cool down value specified for the Auto Scaling group.
- *scaling_adjustment*: the amount by which to scale, based on the specified adjustment type. A positive value adds to the current capacity while a negative number removes from the current capacity.

```
scaleup_policy:
  type: OS::Heat::ScalingPolicy
  properties:
    adjustment_type: change_in_capacity
    auto_scaling_group_id: { get_resource: web_scaler }
    cooldown: {get_param: cooldown_policy_seconds}
    scaling_adjustment: 1

scaledown_policy:
  type: OS::Heat::ScalingPolicy
  properties:
    adjustment_type: change_in_capacity
    auto_scaling_group_id: { get_resource: web_scaler }
    cooldown: {get_param: cooldown_policy_seconds}
    scaling_adjustment: -1
```

- 1.4. The **cpu_alarm_high** and **cpu_alarm_low** resources defines the alarming service alarm. The *alarm_actions* property defines the Auto Scaling policy to invoke when the alarm is triggered.

```
cpu_alarm_high:
  type: OS::Aodh::GnocchiAggregationByResourcesAlarm
  properties:
    description: Scale-up if the average CPU > 30% for 1 minute
```

```

resource_type: instance
metric: cpu_util
aggregation_method: mean
comparison_operator: gt
evaluation_periods: 1
granularity: 60
threshold: 30
alarm_actions:
  - str_replace:
      template: trust+url
      params:
        url: {get_attr: [scaleup_policy, signal_url]}
query:
  str_replace:
    template: '{"=". {"server_group": "stack_id"}}'
    params:
      stack_id: {get_param: "OS::stack_id"}

cpu_alarm_low:
  type: OS::Aodh::GnocchiAggregationByResourcesAlarm
  properties:
    description: Scale-down if the average CPU < 15% for 1 minute
    resource_type: instance
    metric: cpu_util
    aggregation_method: mean
    comparison_operator: lt
    evaluation_periods: 1
    granularity: 60
    threshold: 15
    alarm_actions:
      - str_replace:
          template: trust+url
          params:
            url: {get_attr: [scaledown_policy, signal_url]}
query:
  str_replace:
    template: '{"=". {"server_group": "stack_id"}}'
    params:
      stack_id: {get_param: "OS::stack_id"}

```

- ▶ 2. Review the parameters passed to the **webserver-autoscale** template using the **environment.yaml** environment file.

The **resource_registry** section defines the custom web server instance created by the **web_server_withfloatingip.yaml** template as the resource for the stack. The custom resource is identified as the **My::Server::Custom::WebServer** resource type alias in the stack.

```

resource_registry:
  My::Server::Custom::WebServer: "web_server_withfloatingip.yaml"
parameters:
  key_name: "example-keypair"
  image_name: "rhel7"
  instance_name: "finance-server"
  instance_flavor: "default"
  public_net: "provider-datacentre"

```

```
private_net: "finance-network1"
private_subnet: "finance-subnet1"
initial_capacity: 1
cooldown_policy_seconds: 60
asg_group_min_size: 1
asg_group_max_size: 2
```

- 3. On **workstation**, source the `/home/student/developer1-finance-rc` identity environment for the **developer1** user.

```
[student@workstation ~]$ source ~/developer1-finance-rc
```

- 4. List the stack created by the lab setup script.

```
[student@workstation ~(developer1-finance)]$ openstack stack list --short
+-----+-----+-----+
| ID           | Stack Name      | Stack Status   |
+-----+-----+-----+
| a841c1bf-f395-4f6c-ab7e-0cd352aae74a | autoscale-stack | CREATE_COMPLETE |
+-----+-----+-----+
```

- 5. List the output parameters of the **autoscale-stack** stack. Verify that the web server is reachable using the URL listed by the `website_url_asg` output parameter.

5.1. View the output parameters of the **autoscale-stack** stack:

```
[student@workstation ~(developer1-finance)]$ openstack stack output list \
autoscale-stack
+-----+-----+
| output_key      | description                               |
+-----+-----+
| website_url_asg | This URL is the "external" URL that can be used to |
|                   | access the web server.                         |
|                   |                                                 |
| web_public_ip_asg | Floating IP address of the web server          |
| web_private_ip_asg | IP address of first web server in private network |
+-----+-----+
```

5.2. List the value of the `website_url_asg` parameter returned by the **autoscale-stack** stack:

```
[student@workstation ~(developer1-finance)]$ openstack stack output show \
autoscale-stack \
website_url_asg
+-----+-----+
| Field      | Value                                |
+-----+-----+
| description | This URL is the "external" URL that can be used |
|             | to access the web server.                      |
|             |                                                 |
| output_key  | website_url_asg                         |
| output_value| [                                         |
|             |   "http://172.25.250.P/"                  |
|             | ]                                         |
|             | 
```

- 5.3. Verify that the website URL returned the `website_url_asg` parameter is reachable. The `website_url_asg` parameter value and the output of the `curl` command may vary.

```
[student@workstation ~](developer1-finance)]$ curl http://172.25.250.P
<h1>You are connected to 172.25.250.P</h1>
<h2>The private IP address is:192.168.1.N</h2>
Red Hat Training
```

- 6. List the alarms created by the **autoscale-stack** stack.



IMPORTANT

Both alarm states show **insufficient data**. This is because the default alarm evaluation interval is set to 60 seconds, and in the classroom environment it takes 15 minutes before the metrics are gathered and computed by the metric service.

```
[student@workstation ~](developer1-finance)]$ openstack alarm list -f json
[student@workstation ~](developer1-finance)]$ openstack alarm list -f json
[
  {
    "name": "autoscale-stack-cpu_alarm_high-b7syzmgsh625",
    "enabled": true,
    "alarm_id": "e3291a6f-b579-4f81-bd33-1258f7cae951",
    "state": "insufficient data",
    "type": "gnocchi_aggregation_by_resources_threshold",
    "severity": "low"
  },
  {
    "name": "autoscale-stack-cpu_alarm_low-j2v3sjjnmrqi",
    "enabled": true,
    "alarm_id": "45c94c15-f4f2-4b03-a9b8-82915128cc6e",
    "state": "insufficient data",
    "type": "gnocchi_aggregation_by_resources_threshold",
    "severity": "low"
  }
]
```

The **autoscale-stack-cpu_alarm_low-j2v3sjjnmrqi** alarm triggers when CPU usage is less than **15%** for one minute. The **autoscale-stack-cpu_alarm_high-b7syzmgsh625** alarm triggers when then CPU usage is more than **30%** for one minute. Both alarms are used by orchestration to scale the stack out and in.

- 7. Use the `openstack` command to display the floating IP address associated with the `finance-server-XX` instance. The value of `XX` is a random value. The example output displays a floating IP address of **172.25.250.P**, but the actual IP address may vary.

```
[student@workstation ~](developer1-finance)]$ openstack server list \
-c Name \
-c Networks
```

Name	Networks
finance-server-XX	finance-network1=192.168.1.N, 172.25.250.P

- 8. Use the **ssh** command to connect to the instance using its public IP address. Log in as **cloud-user**.

```
[student@workstation ~](developer1-finance)]$ ssh cloud-user@172.25.250.P
Warning: Permanently added '172.25.250.P' (ECDSA) to the list of known hosts.
[cloud-user@finance-server-XX ~]$
```

- 9. The template file installed a script named **scaling_launchme.sh**. The script stresses the server and triggers the alarming service alarm that monitors for high CPU usage. Run the script.

```
[cloud-user@finance-server-XX ~]$ ./scaling_launchme.sh
Stressing the server...
```

- 10. From **workstation**, open a new terminal and source the **/home/student/developer1-finance-rc** environment file. Monitor the alarms using the **watch** command. Wait for the **autoscale-stack-cpu_alarm_high-1ggn7h4allsx** alarm to transition its state from **ok** to **alarm**.



IMPORTANT

In the classroom environment, the metric service takes 10 minutes to gather and compute metering samples. The alarm state transitions from the **insufficient data** state to the **alarm** or **ok** state after 10 minutes.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~](developer1-finance)]$ watch -d -n 60 \
openstack alarm list -f json
Every 60.0s: openstack alarm list -f json                                Tue Jul  3 18:42:22 2018

[
  {
    "name": "autoscale-stack-cpu_alarm_high-b7syzmsh625",
    "enabled": true,
    "alarm_id": "e3291a6f-b579-4f81-bd33-1258f7cae951",
    "state": "alarm",
    "type": "gnocchi_aggregation_by_resources_threshold",
    "severity": "low"
  },
  {
    "name": "autoscale-stack-cpu_alarm_low-j2v3sjjnmrqi",
    "enabled": true,
    "alarm_id": "45c94c15-f4f2-4b03-a9b8-82915128cc6e",
    "state": "ok",
    "type": "gnocchi_aggregation_by_resources_threshold",
    "severity": "low"
  }
]
```

Press **Ctrl+C** to stop the **watch** command.

- ▶ 11. Wait one or two minutes, then use **watch** with the **openstack** command to monitor the instances. The orchestration engine automatically scales out the resources by launching one new instance.

```
[student@workstation ~](developer1-finance)]$ watch -d -n 5 \
openstack server list -c Name -c Status
Every 5.0s: openstack server list -c Name -c Status      Tue Jul  3 18:52:34 2018

+-----+-----+
| Name | Status |
+-----+-----+
| finance-server-XX | BUILD |
| finance-server-XX | ACTIVE |
+-----+-----+
```

Press **Ctrl+C** to stop the **watch** command.

- ▶ 12. When the background process created by the script stops running, orchestration engine scales in the resources by deleting one of the instances that was created.

Return to the terminal session connected to the **finance-server-XX** instance. Run the **ps** command to ensure that the process called **yes** is no longer running. When done, exit from the instance.

```
[cloud-user@finance-server-XX ~]$ ps -ef | grep yes
cloud-u+ 11331 11308  0 10:05 pts/0    00:00:00 grep --color=auto yes
[cloud-user@finance-server-XX ~]$ exit
```

- ▶ 13. When the **yes** process ends, return back to the **workstation** terminal with the **openstack** command running. Watch the output as the two instances that were created are being deleted. After one instance has been deleted, press **Ctrl+C** to exit the **watch** command.



IMPORTANT

Terminating the instance after the **autoscale-stack-cpu_alarm_low** alarm transitions to the **alarm** state takes approximately 10 minutes in classroom environment. Wait until the second instance has been terminated before cleaning up the exercise.

```
[student@workstation ~](developer1-finance)]$ watch -d -n 5 \
openstack server list -c Name -c Status
Every 5.0s: openstack server list -c Name -c Status      Tue Jul  3 19:03:20 2018

+-----+-----+
| Name | Status |
+-----+-----+
| finance-server-XX | ACTIVE |
```

```
+-----+-----+
```

Press **Ctrl+C** to stop the **watch** command.

- 14. Review the state of the alarms.

```
[student@workstation ~](developer1-finance)]$ openstack alarm list
[
  {
    "name": "autoscale-stack-cpu_alarm_high-b7syzmsh625",
    "enabled": true,
    "alarm_id": "e3291a6f-b579-4f81-bd33-1258f7cae951",
    "state": "ok",
    "type": "gnocchi_aggregation_by_resources_threshold",
    "severity": "low"
  },
  {
    "name": "autoscale-stack-cpu_alarm_low-j2v3sjjnmrqi",
    "enabled": true,
    "alarm_id": "45c94c15-f4f2-4b03-a9b8-82915128cc6e",
    "state": "alarm",
    "type": "gnocchi_aggregation_by_resources_threshold",
    "severity": "low"
  }
]
```

- 15. Review the history of the **autoscale-stack-cpu_alarm_low-j2v3sjjnmrqi** alarm. View the alarm history using the alarm ID of the **autoscale-stack-cpu_alarm_low-j2v3sjjnmrqi** alarm.

```
[student@workstation ~](developer1-finance)]$ openstack alarm-history search \
--query alarm_id="45c94c15-f4f2-4b03-a9b8-82915128cc6e" \
-f json
[
  ...output omitted...
  {
    "alarm_id": "45c94c15-f4f2-4b03-a9b8-82915128cc6e",
    "type": "state transition",
    "detail": "{\"transition_reason\": \"Transition to alarm due to 1 samples outside threshold, most recent: 0.124166956\", \"state\": \"alarm\"}",
    "timestamp": "2018-07-03T14:19:22.704476"
  },
  ...output omitted...
]
```

Cleanup

From **workstation**, run the **lab stacks-autoscale cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab stacks-autoscale cleanup
```

This concludes the guided exercise.

DEPLOYING SCALABLE STACKS

PERFORMANCE CHECKLIST

In this lab, you will use orchestration templates to deploy a web stack with a load balancer.

OUTCOMES

You should be able to:

- Inspect a template to determine required parameters.
- Deploy an orchestration stack to create a load balancer.
- Add web servers behind a load balancer using an orchestration stack.

Ensure that the **workstation** and overcloud's virtual machines are started.

On **workstation**, run the **lab stacks-lab setup** command, which ensures that the required resources are available.

```
[student@workstation ~]$ lab stacks-lab setup
```

1. On **workstation**, download the orchestration template file from <http://materials.example.com/heat/load-balancer.yaml>. Save the template as **/home/student/load-balancer.yaml** and inspect its contents.
2. Source the **/home/student/operator1-production-rc** identity environments file for the **operator1** user.
Use the **openstack** command to retrieve information about the environment. You will use this information in the next step to populate the orchestration environments file to be used by the **/home/student/load-balancer.yaml** template.
3. After you have inspected the resources, create the required environment file, **/home/student/lb-create-environment.yaml**. Use **provider-datacentre** for the public network, **production-subnet1** for the private subnet, and **production-webpool** for the load balancer pool.
4. Launch the stack using the **/home/student/load-balancer.yaml** template file and the **/home/student/lb-create-environment.yaml** environment file. Set the name of the stack to **production-lb-stack**.
5. Download the orchestration templates for the second stack, **production-webstack**. The stack requires two files, **web_server_withoutfloatingip.yaml** and **lb-add-members.yaml**, located at <http://materials.example.com/heat>. Save the two files in the **/home/student** directory.
6. Create the environment file for the **production-webstack** stack as **/home/student/lb-members-environment.yaml**. Include the following resources in the environment file:
Include the resources from the stack defined in the **web_server_withoutfloatingip.yaml** template file. Specify the type as **My::Server::Custom::WebServer**.

Parameters for production-webstack

PARAMETER NAME	PARAMETER VALUE
key_name	example-keypair
image_name	rhel7-web
instance_name	production-server
instance_flavor	default
public_net	provider-datacentre
private_net	production-network1
private_subnet	production-subnet1
pool_name	production-webpool
servercount	2

7. Launch the second stack, **production-webstack**, with the `/home/student/lb-members-environment.yaml` environment file and the `/home/student/lb-add-members.yaml` template file.
8. Verify that two instances are launched behind the load balancer.
Retrieve the public IP address attached to the load balancer launched, using the `website_url` output parameter of the **production-lb-stack** stack.
9. Verify that the load balancer distributes the incoming HTTP requests between the two web servers. Use the public IP address associated with the load balancer.

Evaluation

On **workstation**, run the `lab stacks-lab grade` command to confirm success of this exercise.

```
[student@workstation ~]$ lab stacks-lab grade
```

Cleanup

On **workstation**, run the `lab stacks-lab cleanup` script to clean up this exercise.

```
[student@workstation ~]$ lab stacks-lab cleanup
```

This concludes the lab.

DEPLOYING SCALABLE STACKS

PERFORMANCE CHECKLIST

In this lab, you will use orchestration templates to deploy a web stack with a load balancer.

OUTCOMES

You should be able to:

- Inspect a template to determine required parameters.
- Deploy an orchestration stack to create a load balancer.
- Add web servers behind a load balancer using an orchestration stack.

Ensure that the **workstation** and overcloud's virtual machines are started.

On **workstation**, run the **lab stacks-lab setup** command, which ensures that the required resources are available.

```
[student@workstation ~]$ lab stacks-lab setup
```

1. On **workstation**, download the orchestration template file from <http://materials.example.com/heat/load-balancer.yaml>. Save the template as **/home/student/load-balancer.yaml** and inspect its contents.
 - 1.1. Download the orchestration template file from <http://materials.example.com/heat/load-balancer.yaml>. Save the template file in the **/home/student** directory.

```
[student@workstation ~]$ wget \
http://materials.example.com/heat/load-balancer.yaml
...output omitted...
2018-07-03 10:45:48 (186 MB/s) - 'load-balancer.yaml' saved [1546/1546]
```

- 1.2. Open the file and inspect the parameters required by the template. The following output highlights the relevant resources that require a value in the environment file.

```
heat_template_version: queens
description: Create a Load Balancer with round-robin algorithm
parameters:
  public_net:
    type: string
  private_subnet:
    type: string
  pool_name:
    type: string
    default: web-pool
```

2. Source the **/home/student/operator1-production-rc** identity environments file for the **operator1** user.

Use the **openstack** command to retrieve information about the environment. You will use this information in the next step to populate the orchestration environments file to be used by the **/home/student/load-balancer.yaml** template.

- 2.1. Source the **operator1** user environment file:

```
[student@workstation ~]$ source ~/operator1-production-rc  
[student@workstation ~(operator1-production)]$
```

- 2.2. Retrieve the name of the public network:

```
[student@workstation ~(operator1-production)]$ openstack network list \  
--external \  
-c Name  
+-----+  
| Name |  
+-----+  
| provider-datacentre |  
+-----+
```

- 2.3. Retrieve the name of the project subnet:

```
[student@workstation ~(operator1-production)]$ openstack subnet list -c Name  
+-----+  
| Name |  
+-----+  
| production-subnet1 |  
| provider-subnet-172.25.250 |  
+-----+
```

3. After you have inspected the resources, create the required environment file, **/home/student/lb-create-environment.yaml**. Use **provider-datacentre** for the public network, **production-subnet1** for the private subnet, and **production-webpool** for the load balancer pool.

Create the **/home/student/lb-create-environment.yaml** file and add the following parameters:

```
parameters:  
  public_net: "provider-datacentre"  
  private_subnet: "production-subnet1"  
  pool_name: "production-webpool"
```

4. Launch the stack using the **/home/student/load-balancer.yaml** template file and the **/home/student/lb-create-environment.yaml** environment file. Set the name of the stack to **production-lb-stack**.

- 4.1. As **operator1**, launch the stack using the **/home/student/load-balancer.yaml** orchestration template file and the **/home/student/lb-create-environment.yaml** environment file.

```
[student@workstation ~(operator1-production)]$ openstack stack create \  
--environment ~/lb-create-environment.yaml \  
--template ~/load-balancer.yaml \  
-
```

production-lb-stack	
Field	Value
id	0a709c1c-37a3-4974-96ee-8496921cb09d
stack_name	production-lb-stack
description	Create a Load Balancer with round-robin algorithm
creation_time	2018-07-03T05:24:39Z
updated_time	None
stack_status	CREATE_IN_PROGRESS
stack_status_reason	Stack CREATE started

- 4.2. The status of the stack changes after a few minutes from the **CREATE_IN_PROGRESS** state to the **CREATE_COMPLETE** state. Verify that the stack status shows **CREATE_COMPLETE**.

```
[student@workstation ~ (operator1-production)]$ openstack stack show \
production-lb-stack -c stack_status
+-----+-----+
| Field | Value |
+-----+-----+
| stack_status | CREATE_COMPLETE |
+-----+-----+
```

5. Download the orchestration templates for the second stack, **production-webstack**. The stack requires two files, **web_server_withoutfloatingip.yaml** and **lb-add-members.yaml**, located at <http://materials.example.com/heat>. Save the two files in the **/home/student** directory.

```
[student@workstation ~ (operator1-production)]$ wget \
http://materials.example.com/heat/web_server_withoutfloatingip.yaml
...output omitted...
[student@workstation ~ (operator1-production)]$ wget \
http://materials.example.com/heat/lb-add-members.yaml
...output omitted...
```

6. Create the environment file for the **production-webstack** stack as **/home/student/lb-members-environment.yaml**. Include the following resources in the environment file:
Include the resources from the stack defined in the **web_server_withoutfloatingip.yaml** template file. Specify the type as **My::Server::Custom::WebServer**.

Parameters for production-webstack

PARAMETER NAME	PARAMETER VALUE
key_name	example-keypair
image_name	rhel7-web
instance_name	production-server
instance_flavor	default

PARAMETER NAME	PARAMETER VALUE
public_net	provider-datacentre
private_net	production-network1
private_subnet	production-subnet1
pool_name	production-webpool
servercount	2

```

resource_registry:
  My::Server::Custom::WebServer: "web_server_withoutfloatingip.yaml"
parameters:
  key_name: "example-keypair"
  image_name: "rhel7-web"
  instance_name: "production-server"
  instance_flavor: "default"
  public_net: "provider-datacentre"
  private_net: "production-network1"
  private_subnet: "production-subnet1"
  pool_name: "production-webpool"
  servercount: 2

```

7. Launch the second stack, **production-webstack**, with the **/home/student/lb-members-environment.yaml** environment file and the **/home/student/lb-add-members.yaml** template file.
- 7.1. Launch the second stack and specify **production-webstack** as the name. Use the **/home/student/lb-members-environment.yaml** environment file and the **/home/student/lb-add-members.yaml** template file.

```
[student@workstation ~ (operator1-production)]$ openstack stack create \
--environment ~/lb-members-environment.yaml \
--template ~/lb-add-members.yaml \
production-webstack
+-----+-----+
| Field      | Value   |
+-----+-----+
| id          | ff76f841-f1dc-40bd-ba37-5ef1254a2dcf |
| stack_name  | production-webstack |
| description  | Custom webservers behind a Octavia LB |
| creation_time | 2018-07-03T06:07:22Z |
| updated_time | None    |
| stack_status | CREATE_IN_PROGRESS |
| stack_status_reason | Stack CREATE started |
+-----+-----+
```

- 7.2. The status of the stack will change after a few minutes, from the **CREATE_IN_PROGRESS** state to the **CREATE_COMPLETE** state. Verify that the stack status shows **CREATE_COMPLETE**.

```
[student@workstation ~ (operator1-production)]$ openstack stack show \
production-webstack -c stack_status
```

Field	Value
stack_status	CREATE_COMPLETE

8. Verify that two instances are launched behind the load balancer.

Retrieve the public IP address attached to the load balancer launched, using the **website_url** output parameter of the **production-lb-stack** stack.

- 8.1. View instances launched by the orchestration template behind the load balancer.

```
[student@workstation ~(operator1-production)]$ openstack server list \
-c Name \
-c Status \
-c Networks
+-----+-----+
| Name | Status | Networks |
+-----+-----+
| production-server0 | ACTIVE | production-network1=192.168.1.R |
| production-server1 | ACTIVE | production-network1=192.168.1.S |
+-----+-----+
```

- 8.2. Retrieve the public IP address attached to the load balancer launched, using the **website_url** output parameter of the **production-lb-stack** stack. The example output displays a public IP address of **172.25.250.P**, but the actual IP address may vary.

```
[student@workstation ~(operator1-production)]$ openstack stack output list \
production-lb-stack
+-----+
| output_key | description |
+-----+
| website_url | This is the URL that is used to access the load balancer. |
+-----+
[student@workstation ~(operator1-production)]$ openstack stack output show \
production-lb-stack website_url
+-----+
| Field | Value |
+-----+
| description | This is the URL that is used to access the load balancer. |
| output_key | website_url |
| output_value | http://172.25.250.P |
+-----+
```

9. Verify that the load balancer distributes the incoming HTTP requests between the two web servers. Use the public IP address associated with the load balancer.

- 9.1. Use the **curl** command twice to access the public URL of the load balancer. Verify that the load balancer distributes the incoming HTTP requests between the two web servers.

```
[student@workstation ~(operator1-production)]$ curl http://172.25.250.P
<h1>You are connected to 192.168.1.R</h1>
Red Hat Training
[student@workstation ~(operator1-production)]$ curl http://172.25.250.P
```

```
<h1>You are connected to 192.168.1.5</h1>
```

Red Hat Training

Evaluation

On **workstation**, run the **lab stacks-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab stacks-lab grade
```

Cleanup

On **workstation**, run the **lab stacks-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab stacks-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The Telemetry service supports many back ends to store metric and alarm data.
- The Orchestration service provides OpenStack with stack deployment orchestration features. Component services include the engine service, the API service, and legacy implementations of the Amazon CloudFormation orchestration service.
- An Orchestration service template declares the infrastructure for a cloud-based application. Templates are human-readable text files that can be maintained in a version control system.
- A Heat Orchestration Template (HOT) is a YAML file with three major sections: **parameters**, **resources**, and **outputs**.
- The **parameters** section defines the list of values that can be assigned to a template. Each parameter has a defined data type and an optional default value to use if a value is not specified when the template is deployed.
- The Orchestration service implements Auto Scaling features. OpenStack can scale instances out and in, based on their utilization.
- Telemetry alarms allow system administrators to determine the conditions under which scaling is triggered.

DEPLOYING AN OPENSTACK OVERCLOUD

GOAL

Deploy an OpenStack proof of concept using the director UI and provisioning service templates.

OBJECTIVES

- Deploy OpenStack using the director UI and provisioning service templates.
- Verify OpenStack deployment functionality by querying services and launching an instance.

SECTIONS

- Creating an Overcloud Deployment Plan Using Director UI (and Guided Exercise)
- Verifying the OpenStack Overcloud Deployment (and Guided Exercise)

LAB

- Deploying an OpenStack Overcloud

CREATING OVERCLOUD DEPLOYMENT PLAN USING DIRECTOR WEB UI

OBJECTIVES

After completing this section, students should be able to deploy an OpenStack overcloud using director web UI and service provisioning templates.

INTRODUCTION TO RED HAT OPENSTACK PLATFORM DIRECTOR

OpenStack core components provide a comprehensive set of services to provision end user cloud workloads consisting of deployed server instances organized by projects. With orchestration, arrangements of complex multi-server applications have become easy to define and deploy with push-button simplicity. To address the complexity of installation and management of OpenStack cloud infrastructure itself, Red Hat OpenStack Platform director is provided as a web UI and a command line interface.

As the default deployment and installation tool, Red Hat OpenStack Platform Director offers the most advanced and flexible method of installation, including automated platform-wide upgrades and updates. It is based primarily on the Deployment service component known as TripleO, an abbreviation for "OpenStack On OpenStack". The Deployment service uses OpenStack components running on a dedicated all-in-one installation (the undercloud) to install an operational OpenStack cloud (the overcloud), utilizing extended core components and additional components to locate, provision, deploy, and configure bare metal systems as OpenStack controller, compute, networking, and storage nodes. Red Hat recommends Red Hat OpenStack Platform Director for all installations, even smaller proof-of-concept and development. Use of director is required for Red Hat OpenStack Platform infrastructure with a Red Hat support contract. The following table describes the OpenStack deployment component services.

OpenStack Component Services for OpenStack-On-OpenStack

SERVICE	FUNCTION
Orchestration for TripleO	Provides a set of YAML-based templates to define configuration and provisioning instructions to deploy OpenStack infrastructure servers. Orchestration, defined previously as a core component, defines server roles to provision OpenStack infrastructure.
Bare Metal Provisioning	Enables provisioning server instance deployments to physical (bare metal) machines using hardware-specific drivers. Bare Metal Provisioning integrates with the Compute service to provision the bare metal machines in the same way as virtual machines, first introspecting the physical machines to obtain hardware attributes and configuration.
Workflow	Managed by the Mistral workflow service. A user typically writes a workflow using workflow language based on YAML and uploads the workflow definition to Mistral with its REST API. Users can start this workflow manually using the same API, or configure a trigger to start the workflow on some event. Provides a set of workflows for certain Red Hat OpenStack Platform director-specific actions, such as importing and deploying plans.

SERVICE	FUNCTION
Messaging	Provides a secure and scalable messaging service for providing asynchronous communication for intra-cloud applications. Other OpenStack components integrate with Messaging to provide functional equivalence to third-party Simple Queue Service (SQS) and Simple Notification Service (SNS) services. Messaging provides communication for the Workflow service.
Deployment	Provides a tool set for installing, upgrading, and operating OpenStack clouds using OpenStack components and methods.

INTRODUCING THE UNDERCLOUD AND OVERCLOUD

The Red Hat OpenStack Platform uses the terms *undercloud* and *overcloud* to distinguish between the standalone Red Hat OpenStack Platform director cloud which deploys and manages production clouds, and the production cloud which deploys and manages project workloads.

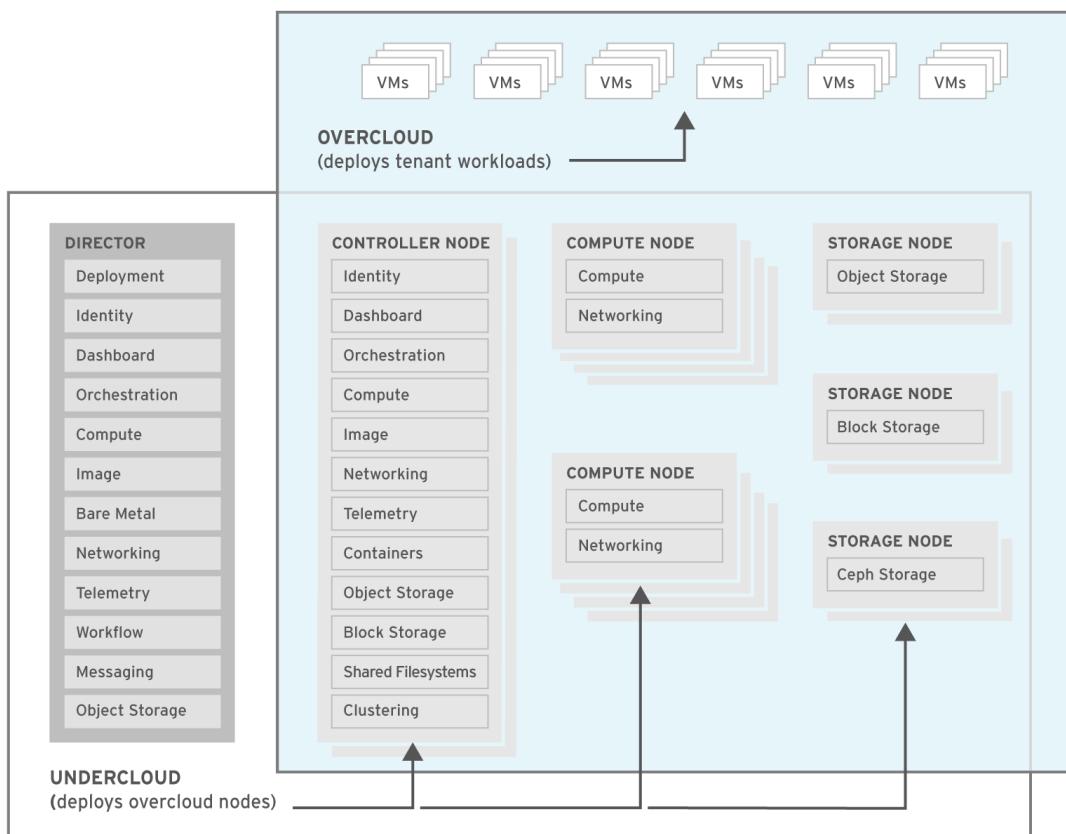


Figure 13.1: An undercloud deploys an overcloud

The undercloud is the Red Hat OpenStack Platform director machine itself, plus the provisioning network and resources required for provisioning and managing the OpenStack nodes that form the overcloud. During the building process for the overcloud, the machine nodes being provisioned to become controller, compute, network, and storage systems are considered to be the workload of the undercloud. When deployment and all configuration stages are complete, these nodes reboot to become the overcloud.

The overcloud is a Red Hat OpenStack Platform environment resulting from a template configuration deployed from the undercloud. Prior to the introduction of the undercloud, any similar Red Hat OpenStack Platform environment would have simply been called *the cloud*.

Using the terms undercloud and overcloud provides a distinction between the two Red Hat OpenStack Platform installations. Each cloud has a complete set of component services, endpoints, authentication, and purpose. To access and manage the undercloud, connect to the Identity service endpoint of the Red Hat OpenStack Platform director system. To access and manage the overcloud, connect to the Identity service endpoint on a controller system in the overcloud.

The undercloud installs the overcloud. However, the undercloud is not only an installation tool set. It is a comprehensive platform for managing, monitoring, upgrading, scaling, and deleting overclouds. Currently, the undercloud supports deploying and managing a single overcloud.

Functions Provided by Undercloud

The undercloud provides multiple functions using the default installed components:

Infrastructure Planning

The undercloud includes a default set of roles such as Compute, Controller, and several others, and provides the ability to use custom roles. These custom roles allow a user to deploy overcloud nodes based on OpenStack services that are required to be deployed. The undercloud allows these roles to be assigned to a specific bare metal node.

Bare Metal System Management

The undercloud uses the Bare Metal provisioning service to first perform introspection on each candidate node, querying and recording node-specific capabilities and configuration. The undercloud uses an out-of-band management interface, usually Intelligent Platform Management Interface (IPMI), to manage, monitor, and configure bare metal systems.

Orchestration of overcloud deployment

The undercloud uses the orchestration service that provides a template-based engine for the undercloud, to create and manage resources such as storage, networking, instances, and applications as a repeatable running environment. These templates act as a set of plans, which are imported into object storage containers before the overcloud deployment. The default orchestration templates and plans are located at **/usr/share/openstack-tripleo-heat-templates**. The plans contain hooks that allow customization based on the overcloud environment.

CLI and Web UI tool sets

The Red Hat OpenStack Platform director provides both CLI and web UI tool sets to perform the above functions using the command-line interface or a web-based user interface.

UNDERSTANDING OVERCLOUD PLANS

An overcloud deployment plan consists of orchestration templates and environment files with deployment instructions. The **plan-environment.yaml** file contains the plan metadata. When creating an overcloud plan, provide the plan name and directory containing the templates and environment as arguments. The plan name is used as the name of the object storage container, the workflow environment, and the orchestration stack for this overcloud. As the overcloud plan is created, the plan files are uploaded to an object store container on the undercloud.

Default orchestration heat templates are located in **/usr/share/openstack-tripleo-heat-templates**. After copying this set to a working directory called **~/demo-templates**, the plan is uploaded from the CLI using the following command:

```
(undercloud)[stack@director ~]$ openstack overcloud plan create --templates ~/demo-templates demo-overcloud
Creating Swift container to store the plan
Creating plan from template files in: /home/stack/demo-templates
Started Mistral Workflow tripleo.plan_management.v1.create_deployment_plan.
Execution ID: 28172b1c-8dd3-42ab-9f64-e8fed805f1c0
```

Plan created.

Use **openstack overcloud plan list** to view existing overcloud plans.

```
(undercloud)[stack@director ~]$ openstack overcloud plan list
+-----+
| Plan Name      |
+-----+
| demo-overcloud |
| overcloud       |
+-----+
```

Use **openstack overcloud plan export** to export an overcloud plan to files. The default export file name is **planname.tar.gz**.

```
(undercloud)[stack@director ~]$ openstack overcloud plan export demo-overcloud
Exporting plan demo-overcloud...
Started Mistral Workflow tripleo.plan_management.v1.export_deployment_plan.
Execution ID: d3a5bf57-4c39-447f-b569-859cec979314
...output omitted...
(undercloud)[stack@director ~]$ ls -l demo-overcloud.tar.gz
-rw-rw-r--. 1 stack stack 500848 Jun 18 12:01 demo-overcloud.tar.gz
```

Composition of an Overcloud Plan

The **plan-environment.yaml** contains the plan metadata with parameters required to deploy the overcloud. To view the **plan-environment.yaml** file from the exported overcloud plan, extract the exported archive.

The default structure of the **plan-environment.yaml** file contains the following:

```
① description: 'Default Deployment Plan'
② environments:
  - path: overcloud-resource-registry-puppet.yaml
  - path: environments/docker.yaml
  - path: environments/docker-ha.yaml
  - path: environments/containers-default-parameters.yaml
③ name: demo-overcloud
④ parameter_defaults:
  RootStackName: demo-overcloud
```

- ① A short description of the plan.
- ② The list of environment files to be included in the plan. For example, to deploy overcloud using containerized services along with high availability the **environments/docker.yaml**, and **environments/docker-ha.yaml** files are used.
- ③ The name of the plan.
- ④ The list of parameters required to deploy the overcloud. For example, the **RootStackName** parameter defines the overcloud orchestration stack name.

OVERCLOUD PLAN TOPOLOGY FOR DEFAULT CLASSROOM

The following Figure 13.2 shows four deployed nodes: **controller0**, **compute0**, **compute1**, and **ceph0**.

Figure 13.2: Classroom topology resulting from the default CL110 course plan

In the classroom environment, the pre-deployed overcloud uses the following parameters in the **plan-environment.yaml** file to define the network infrastructure and quantity for each overcloud node role:

- The file includes the overall network configuration for the overcloud. For example, the IP addressing used for the Controller IPs.

```
(undercloud) [stack@director overcloud]$ cat plan-environment.yaml \
| grep -A 12 'ControllerIPs'
ControllerIPs:
  external:
    - 172.25.250.1
  internal_api:
    - 172.24.1.1
  management:
    - 172.24.5.1
  storage:
    - 172.24.3.1
  storage_mgmt:
    - 172.24.4.1
  tenant:
    - 172.24.2.1
```

- An overcloud can isolate various network traffic on a single network interface using different VLAN IDs.

```
(undercloud) [stack@director overcloud]$ cat plan-environment.yaml | grep "VlanID"
InternalApiNetworkVlanID: 10
ManagementNetworkVlanID: 50
StorageMgmtNetworkVlanID: 40
StorageNetworkVlanID: 30
TenantNetworkVlanID: 20
```

- IP addresses for overcloud nodes can be set persistently, both for node interfaces and for the high availability Virtual IPs (VIPs). For example, this output lists the public virtual fixed IP:

```
(undercloud) [stack@director overcloud]$ cat plan-environment.yaml | sed '1d' |
grep "PublicVirtualFixedIPs"
PublicVirtualFixedIPs:
  - ip_address: 172.25.250.50
```

- The plan file defines the quantity of nodes for each overcloud role.

```
(undercloud) [stack@director overcloud]$ cat plan-environment.yaml | grep "Count"
CephStorageCount: 1
ComputeCount: 2
ControllerCount: 1
```

A containerized overcloud deploys various services into service containers. OpenStack services may map to multiple containers, not just one-to-one to individual containers. These containerized OpenStack services are comprised of multiple discrete containerized processes.

To deploy a containerized overcloud, the overcloud deployment requires access to a container registry with the required container images. Red Hat OpenStack Platform supports three registry types based on where images are set to be pulled from: **Remote registry**, **Local registry**, and **Satellite server**. In the classroom environment, the local container registry is configured on the **director** node.

The **plan-environment.yaml** file contains the destination of the these container images:

```
(undercloud) [stack@director overcloud]$ cat plan-environment.yaml | grep "Docker"
DockerAodhApiImage: 172.25.249.200:8787/rhosp13/openstack-aodh-api:latest
DockerAodhConfigImage: 172.25.249.200:8787/rhosp13/openstack-aodh-api:latest
DockerAodhEvaluatorImage: 172.25.249.200:8787/rhosp13/openstack-aodh-
evaluator:latest
DockerAodhNotifierImage: 172.25.249.200:8787/rhosp13/openstack-aodh-
notifier:latest
DockerCephDaemonImage: 172.25.249.200:8787/rhceph/rhceph-3-rhel7:latest
...output omitted...
```

DEPLOYING OVERCLOUD USING THE DIRECTOR WEB UI

The Red Hat OpenStack Platform director web UI provides users with a web-based tool to deploy an overcloud. Using the director web UI is recommended for many use cases, including:

- To deploy an overcloud for proof-of-concept.
- To demonstrate validations that the resulting deployment meets customer needs.
- To create a deployment plan to be used repeatedly to deploy similar overclouds.

Configure the director web UI and services when installing the undercloud. To enable the UI, set the following **/home/stack/undercloud.conf** parameter on **director** before installing.

```
(undercloud) [stack@director ~]$ grep enable_ui undercloud.conf
enable_ui = true
```

Accessing the Director Web UI

The web UI is accessible using the undercloud IP address. Accessing the director web UI from remote hosts may require creating SSH tunnels to reach the undercloud endpoints. In the classroom environment, access the web UI at <https://director-ui.lab.example.com>.

To log in, provide the administrator user name and password; default **admin**. Retrieve the current password for the administrator user by querying the installation data:

```
[user@demo ~]$ ssh stack@director sudo hiera admin_password
redhat
```

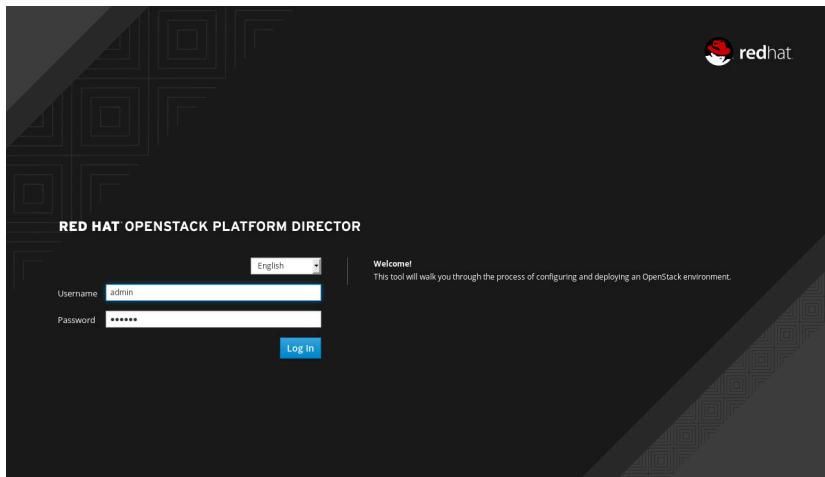


Figure 13.3: director web UI login screen

IMPORTING AN OVERCLOUD PLAN USING DIRECTOR WEB UI

The director web UI allows user to upload an overcloud plan from either an archived and compressed gzip file, or unpacked in a directory.

Uploading an Overcloud Plan using Director Web UI

The following steps outline the process to upload an overcloud plan using the director web UI:

1. From the Red Hat OpenStack Platform director dashboard, click Import Plan in the Plans tab.
2. In the New Plan tab, enter the name of the plan in the Plan Name text box.
3. Select the Upload Type as Local Folder or Tar Archive (.tar.gz or .tgz).
4. Click the Browse button to select the Plan Files. Browse and select the directory or the archive file. Click Upload.
5. Click Upload Files and Create Plan. Wait until the overcloud plan files are uploaded and the plan is created on the **director** node. When complete, a notification box appears on the dashboard.

Figure 13.4: Import an overcloud plan

Inspecting Bare Metal Nodes using Director Web UI

The director web UI can register new bare metal nodes. The Nodes tab at the top of the web UI provides methods for registering new nodes and introspecting registered nodes. This section of the page also provides information such as the power state, introspection status, provision state,

and hardware information of the bare metal node. The bare metal node must be in the **available** state for the node to be included in the deployment.

Running OpenStack Validations using Director Web UI

The Red Hat OpenStack Platform ships collection of Ansible playbooks to make the OpenStack validations easier. On **director**, validations are in the *openstack-tripleo-validations* package installed during the undercloud deployment. These validation Ansible playbooks are categorized by deployment phases, such as *prep*, *pre-deployment*, or *post-deployment*. Execute these validations using the director web UI or the OpenStack CLI. The playbooks are stored in the **/usr/share/openstack-tripleo-validations/validations** on **director**.

Click on the button next to Logout to open the Validations screen. To run a validation, click the play button for a listed validation. Click a validation link to open the Validation Details. A red symbol indicates a failed validation, while a green check indicates a successful validation.

MANAGING AN OVERCLOUD PLAN USING DIRECTOR WEB UI

The director web UI allows users to customize the plan for hardware and environment requirements. Once the plan is uploaded, the plan appears in the Plan listing. To manage a plan, click on the plan card. The plan screen provides user with four main steps to deploy an overcloud:

1. **Prepare Hardware:** Click Register Nodes to add new bare metal nodes. If the bare metal nodes are already registered, **Prepare Hardware** can be skipped.
2. **Specify Deployment Configuration:** Click Edit Configuration to modify the base overcloud configuration. This provides a method to customize the features required from your overcloud. For example, choose various storage options supported by the overcloud such as NFS, iSCSI, Ceph, or proprietary storage products. You can modify various environment parameters based on the environment. Click Save Changes to save the modification.
3. **Configure Roles and Assign Nodes:** This step assigns bare metal nodes into roles from the overcloud plan. You can assign a quantity of nodes to a role by using the spinner widget for each role.
4. **Deploy:** Click Validate and Deploy to deploy the overcloud plan. A warning message appears if pre-deployment validations did not pass. Click Deploy to start the deployment. The View detailed information link displays the overall progress and resources deployed by the orchestration stack.

Editing an Overcloud Plan using Director Web UI

The following steps outline the process to edit an overcloud plan using the director web UI:

1. Log in to the director web UI using the administrator's account.
2. Click the plan card.
3. In the Plans tab, under the Specify Deployment Configuration section click Edit Configuration.
4. In the Deployment Configuration screen, use the Overall Settings tab to include or remove different features from the overcloud deployment.
Click Save Changes.
5. In the Deployment Configuration screen, use the Parameters tab to edit parameters included from the base-level and user provided environment file.
Click Save and Close.

VALIDATING AND DEPLOYING OVERCLOUD USING DIRECTOR WEB UI

Once the overcloud plan is edited and configured, click Validate and Deploy to start the overcloud deployment. An error with the **Not all pre-deployment validations have passed** message appears. This indicates that the new Ansible validations have one or more failures or still need to run. You can deploy without waiting for validations or to ignore validation failures. Click Deploy, and the UI requests that a deployment begin.

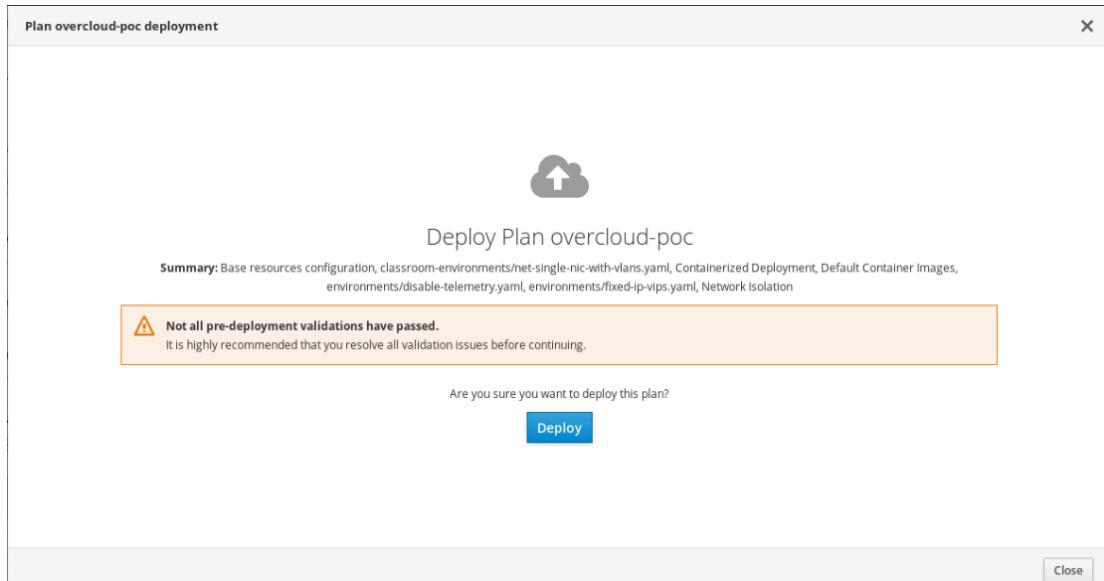


Figure 13.5: Not all pre-deployment validations have passed

When deployment is complete, the IP address, overcloud administrator user name, and password are displayed in the Deploy section.

4 Deploy ⓘ

Deployment succeeded
Stack CREATE completed successfully

Overcloud information:

- Overcloud IP address: 172.25.250.50
- Username: admin
- Password: redhat

[Delete Deployment](#)

Figure 13.6: Information to access the overcloud after deployment



REFERENCES

Additional information is available in the section on *Configuring a Basic Overcloud with the Web UI* in the *Director Installation and Usage* for Red Hat OpenStack Platform, which can be found at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/director_installation_and_usage

Further information is available about *RHOSP Director at TripleO Architecture*

<https://docs.openstack.org/tripleo-docs/latest/install/introduction/architecture.html>

Further information is available in the *Red Hat OpenStack Platform Director Life Cycle* at

<https://access.redhat.com/support/policy/updates/openstack/platform/director>

CREATING AN OVERCLOUD DEPLOYMENT PLAN USING DIRECTOR WEB UI

In this exercise, you will use director web UI to generate a deployment plan and deploy Red Hat OpenStack Platform.

OUTCOMES

You should be able to:

- Export an overcloud plan and review the plan environment file.
- Import an overcloud plan to the Red Hat OpenStack Platform director using the web UI.
- Edit deployment configurations in the imported overcloud plan.

Confirm that the **workstation** and the overcloud's virtual machines are started.

Log in to **workstation** as **student** using **student** as the password. Run the **lab create-overcloudplan setup** command to ensure the **overcloud** plan exists.

```
[student@workstation ~]$ lab create-overcloudplan setup
```

- 1. From **workstation**, log in to the **director** virtual machine using the **stack** user. Export the **overcloud** plan.
- 1.1. Using SSH, log in to the **director** virtual machine as **stack** user.

```
[student@workstation ~]$ ssh stack@director  
(undercloud) [stack@director ~]$
```

- 1.2. Use the **openstack overcloud plan list** to list the plans.

```
(undercloud) [stack@director ~]$ openstack overcloud plan list  
+-----+  
| Plan Name |  
+-----+  
| overcloud |  
+-----+
```

- 1.3. Use the **openstack overcloud plan export** to export the **overcloud** plan.

```
(undercloud) [stack@director ~]$ openstack overcloud plan export overcloud  
Exporting plan overcloud...  
Started Mistral Workflow tripleo.plan_management.v1.export_deployment_plan.  
Execution ID: 5700...0f91
```

```
https://172.25.249.201:13808/v1/AUTH_.../plan-exports/overcloud.tar.gz?  
temp_url_sig=...
```

- 1.4. Log out of the **director** virtual machine.

```
(undercloud) [stack@director ~]$ logout  
Connection to director closed.  
[student@workstation ~]$
```

- ▶ 2. On **workstation**, copy the exported archive to the **/home/student/overcloudplan** directory. Extract the exported archive in the **/home/student/overcloudplan** directory.
 - 2.1. Create a directory named **overcloudplan** in the student's home directory. Copy the **/home/stack/overcloud.tar.gz** file from the **director** to **workstation** in the **/home/student/overcloudplan** directory.

```
[student@workstation ~]$ mkdir ~/overcloudplan  
[student@workstation ~]$ cd ~/overcloudplan  
[student@workstation overcloudplan]$ scp stack@director:~/overcloud.tar.gz .  
...output omitted...
```

- 2.2. Extract the **/home/stack/overcloud.tar.gz** file in the **/home/student/overcloudplan** directory.

```
[student@workstation overcloudplan]$ tar xzvf overcloud.tar.gz  
...output omitted...
```

- ▶ 3. Review the **/home/student/overcloudplan/plan-environment.yaml** environment file that defines networks and VLANs for each overcloud node. Also review the location of the container images.
 - 3.1. Review the **/home/student/overcloudplan/plan-environment.yaml** environment file defining the networks and VLANs for each overcloud node. For example, this partial output lists the IP addresses and VLANs defined to isolate network traffic on a controller node:

```
[student@workstation overcloudplan]$ grep -A12 -e "ControllerIPs" \  
plan-environment.yaml  
ControllerIPs:  
  external:  
    - 172.25.250.1  
  internal_api:  
    - 172.24.1.1  
  management:  
    - 172.24.5.1  
  storage:  
    - 172.24.3.1  
  storage_mgmt:  
    - 172.24.4.1  
  tenant:  
    - 172.24.2.1  
[student@workstation overcloudplan]$ grep -e "VlanID" plan-environment.yaml  
InternalApiNetworkVlanID: 10  
ManagementNetworkVlanID: 50
```

```
StorageMgmtNetworkVlanID: 40
StorageNetworkVlanID: 30
TenantNetworkVlanID: 20
```

- 3.2. Review the **/home/student/overcloudplan/plan-environment.yaml** file to view the location of the container images. For example, the container image used by the **horizon** container is configured for the **director** node.

```
[student@workstation overcloudplan]$ grep -e "DockerHorizonImage" \
plan-environment.yaml
DockerHorizonImage: 172.25.249.200:8787/rhosp13/openstack-horizon:latest
```

- 4. Create the **/home/student/overcloud-poc** overcloud plan directory on **workstation** with the director's orchestration templates.
- 4.1. Use the **scp** command to copy the **/usr/share/openstack-tripleo-heat-templates** directory from the **director** virtual machine to **/home/student/overcloud-poc** on **workstation**.

```
[student@workstation overcloudplan]$ scp -r \
stack@director:/usr/share/openstack-tripleo-heat-templates ~/overcloud-poc
...output omitted...
```

- 5. Use the **https://director-ui.lab.example.com** URL to access the director web UI. Use the administrator user name and password to access the web UI. The default administrator user name is **admin**.
- 5.1. From **director**, use the **sudo hiera admin_password** command to retrieve the password for the default user, **admin**.

```
[student@workstation overcloudplan]$ ssh stack@director sudo hiera \
admin_password
redhat
```

- 5.2. Open the **https://director-ui.lab.example.com** URL to access the director web UI. If a certificate error appears, accept the self-signed certificate.
- 5.3. Log in using **admin** as user name and **redhat** as the password.

- 6. In the Red Hat OpenStack Platform director web UI, import an overcloud plan by uploading the files stored in the **/home/student/overcloudplan** directory. Enter the overcloud plan name as **overcloud-poc**.
- 6.1. From the director dashboard, click Import Plan in the Plans tab.
 - 6.2. In the New Plan tab, enter **overcloud-poc** as the Plan Name.
 - 6.3. Select the Upload Type as Local Folder.
 - 6.4. Click the Browse button to select the Plan Files. Browse and select the **overcloud-poc** directory in the **student** home directory. Click Upload.
 - 6.5. Click Upload Files and Create Plan. Wait until the overcloud plan files are uploaded and the plan is created. When completed, a notification box appears on the dashboard.

- ▶ 7. In the director web UI, edit the **overcloud-poc** deployment plan to enable the creation of isolated OpenStack network VLANs on a single NIC.
 - 7.1. From the Plans tab, click the **overcloud-poc** plan card. The web page shows four steps to deploy the **overcloud-poc** plan.
 - 7.2. Under the Specify Deployment Configuration section, click Edit Configuration to modify the base overcloud configuration.
 - 7.3. In the Deployment Configuration screen, the Overall Settings tab is used to select plan features. Click Network Configuration. Select Network Isolation from the Network Isolation section. Select Single NIC with Vlans No External Ports from the NICs, Bonding, VLANs Configuration section. Click Save Changes.
- ▶ 8. The **overcloud-poc** base plan deploys on one controller and one compute node. A minimum of three controller nodes are required for high availability. Edit the **overcloud-poc** deployment plan configuration to disable high availability configuration for an overcloud controller node.

In the Overall Settings tab, click General Deployment Options. Deselect HA services via Docker in the High Availability section. Click Save Changes. Click Save Changes.
- ▶ 9. In the Deployment Configuration screen, set the default route of the control plane network.

In the Deployment Configuration screen, the Parameters tab provides a method to change various environment parameters. Click Single NIC with Vlans No External Ports. In the ControlPlaneDefaultRoute text box, enter **172.25.249.200** to match the classroom network. Click Save Changes.
- ▶ 10. In the Parameters tab, review the default container image.

Click Default Container Images. View the container image location. The container image for various OpenStack services points to Red Hat container catalog. Click Cancel.
- ▶ 11. In the Configure Roles and Assign Nodes section, ensure that two nodes are selected. One of the nodes is assigned the **Controller** role. The other is assigned the **Compute** role.
- ▶ 12. **Do not** deploy the **overcloud-poc** overcloud plan. There is already an overcloud plan deployed named **overcloud**. The Red Hat OpenStack Platform director currently does not support deployment of multiple overclouds.

Click Logout.

Cleanup

From **workstation**, run the **lab create-overcloudplan** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab create-overcloudplan cleanup
```

This concludes the guided exercise.

VERIFYING THE OPENSTACK OVERCLOUD DEPLOYMENT

OBJECTIVES

After completing this section, students should be able to verify the OpenStack deployment functionality by testing the services and launching an instance.

VERIFYING AN OVERCLOUD DEPLOYMENT

Once built, an overcloud is a production infrastructure with many interacting components. Red Hat OpenStack Platform Director can deploy the Red Hat OpenStack Platform services on one or more machines. In the environment in this course, four virtual machines have been used to deploy the overcloud, one controller node, two compute nodes, and a storage node. The Red Hat OpenStack Platform core services, such as the image service or the block storage service, are deployed on the controller node. The compute node is configured with the required services for acting as a hypervisor. The director is capable of autoscaling or replacing HA controllers and compute nodes.

Currently, the undercloud allows only a single overcloud at a time. However, the OpenStack workflow service can manage multiple overcloud plans and stacks. In a future release, the undercloud will be able to install, access, and manage multiple overclouds.

In current version of Red Hat OpenStack Platform the majority of the OpenStack services now run in isolated service containers. Use the **docker** command to manage these service containers.

Viewing orchestration results

Orchestration deployed each registered nodes as one of the default server roles. Compare the orchestration templates and environment files to the finished nodes. To log in to overcloud nodes, use the **heat-admin** user, the same Linux user account used by Orchestration to access and configure the systems using SSH. From **director**, the **stack** user has passwordless SSH access. The **heat-admin** user has sudo privileges, so use **sudo -i** to switch to **root**. View the following resources to verify the overcloud configuration:

- Compare the NIC configuration of the controller deployment role network orchestration template to the network interfaces and Open vSwitch bridges on controller0.
- Compare the NIC configuration of the compute deployment role network orchestration template to the network interfaces on compute0.
- Use the **docker ps** command to list services on each node and verify service status.
- View logs in **/var/log/containers** on each node to check for errors.

Accessing the admin environment file

The director web UI, after a successful deployment, displays the **admin** user account and password, and the dashboard URL. A user environment file to access a deployed overcloud can be downloaded from the dashboard or created manually. To download the environment file from the dashboard, use the following steps:

1. Log in to the dashboard using the **admin** user's credentials provided by the director web UI.
2. From the Project tab, navigate to Project → API Access.
3. In the API Access page, navigate to Download OpenStack RC File → OpenStack RC File (Identity API v3).
4. Save the environment file.

Verifying the OpenStack services

Use the `docker ps` to verify and monitor the OpenStack services. The section Verifying OpenStack Services demonstrates how to use the `docker` command to verify and manage OpenStack services. As seen in previous chapters, much information is available with `openstack` commands. For example, review how many hypervisors were deployed using the `openstack hypervisor list` command:

```
[user@demo ~(admin)$ openstack hypervisor list
+-----+-----+-----+-----+
| ID | Hypervisor Hostname           | Hypervisor Type | Host IP      | State |
+-----+-----+-----+-----+
| 1  | compute1.overcloud.example.com | QEMU          | 172.24.1.12  | up   |
| 2  | compute0.overcloud.example.com | QEMU          | 172.24.1.2   | up   |
+-----+-----+-----+-----+
```

Use the command of a given service, such as the `openstack network` command for the OpenStack networking service, to display the OpenStack networking agents and their status:

```
[user@demo ~(admin)$ openstack network agent list \
-c "Agent Type" -c "Host" -c "Alive" -c "State"
+-----+-----+-----+
| Agent Type     | Host          | Alive | State |
+-----+-----+-----+
| Open vSwitch agent | compute1.overcloud.example.com | :-) | UP   |
| DHCP agent      | controller0.overcloud.example.com | :-) | UP   |
| Metadata agent   | controller0.overcloud.example.com | :-) | UP   |
| Open vSwitch agent | compute0.overcloud.example.com | :-) | UP   |
| L3 agent         | controller0.overcloud.example.com | :-) | UP   |
| Open vSwitch agent | controller0.overcloud.example.com | :-) | UP   |
+-----+-----+-----+
```

Viewing the OpenStack service logs

In Red Hat OpenStack Platform, service containers bind mount the service log file to make them accessible from their physical location in the controller or compute node's host operating system. These log files are located in the `/var/log/containers` and `/var/log/containers/httpd` directories. The OpenStack API service runs as a Web Server Gateway Interface (WSGI) service inside the `httpd` service. To avoid conflict from other services, the log files are stored in the `/var/log/containers/httpd` file. This example shows the Compute service log files present on the `controller0` node:

```
[heat-admin@controller0 ~]$ ls -l /var/log/containers/* /var/log/containers/
httpd/* \
| grep nova
drwxr-xr-x. 2 root root 86 Jun 22 15:31 nova-api
drwxr-xr-x. 2 root root 88 Jun 22 15:24 nova-placement
/var/log/containers/httpd/nova-api:
-rw-r--r--. 1 root root 1164783 Jun 25 11:21 nova_api_wsgi_access.log
-rw-r--r--. 1 root root 564 Jun 25 04:29 nova_api_wsgi_error.log
/var/log/containers/httpd/nova-placement:
/var/log/containers/nova:
-rw-r--r--. 1 42436 42436 3740211 Jun 25 11:21 nova-api.log
-rw-r--r--. 1 42436 42436 2355650 Jun 25 11:21 nova-api-metadata.log
-rw-r--r--. 1 42436 42436 17875 Jun 25 04:30 nova-conductor.log
-rw-r--r--. 1 42436 42436 15281 Jun 25 04:30 nova-consoleauth.log
```

```
-rw-r--r--. 1 42436 42436 83651 Jun 22 15:25 nova-manage.log  
-rw-r--r--. 1 42436 42436 1240 Jun 25 04:28 nova-novncproxy.log  
-rw-r--r--. 1 42436 42436 986177 Jun 25 11:20 nova-placement-api.log  
-rw-r--r--. 1 42436 42436 98249 Jun 25 11:20 nova-scheduler.log
```

Launching an instance to smoke test

Administrators can launch instances in their environment to ensure a proper installation. The deployment of an instance involves the following services:

- Compute service
- Image service
- Identity service
- OpenStack networking service

Verifying the OpenStack Overcloud Deployment

The following steps outline the process for verifying the deployment of the OpenStack Overcloud.

1. Verify that the orchestration stack shows **CREATE_COMPLETE** status and overcloud nodes are in **ACTIVE** state.
2. Log in to the controller and compute nodes to view the OpenStack services running on each node, using the **heat-admin** account that was provisioned during deployment.
3. Create a minimal environment containing the required networks, an image, and a key pair, then launch an instance.
4. After associating a floating IP, verify that the instance is reachable over SSH.

VERIFYING THE OPENSTACK OVERCLOUD DEPLOYMENT

In this exercise, you will verify the functionality of the OpenStack deployment by querying the services and launching an instance.

OUTCOMES

You should be able to:

- Connect to the undercloud node and list overcloud nodes.
- Connect to an overcloud node and verify the status of installed OpenStack services.
- Create a base environment comprised of an image, a network environment, and a key pair.
- Launch an instance in the environment.

Confirm that the **workstation** and overcloud's virtual machines are started.

Log in to workstation as **student** using **student** as the password. Run the **lab verify-overcloud setup** command to ensure that the OpenStack services are running.

```
[student@workstation ~]$ lab verify-overcloud setup
```

- ▶ 1. Use SSH to connect as the **stack** user to **director**. Review the state of the orchestration stack. List the overcloud virtual machines and ensure that the status is **ACTIVE**.
- 1.1. From **workstation**, open a terminal to connect as the **stack** user to **director**.

```
[student@workstation ~]$ ssh stack@director  
(undercloud) [stack@director ~]$
```

- 1.2. Run the **openstack stack list** command to list the orchestration stack. Ensure that the state of the **overcloud** stack is **CREATE_COMPLETE**.

```
(undercloud) [stack@director ~]$ openstack stack list \  
-c "Stack Name" -c "Stack Status"  
+-----+-----+  
| Stack Name | Stack Status |  
+-----+-----+  
| overcloud | CREATE_COMPLETE |  
+-----+-----+
```

- 1.3. List the overcloud nodes and the status. Ensure the status of all overcloud nodes are **ACTIVE**.

```
(undercloud) [stack@director ~]$ openstack server list \  
-c "Name" -c "Status"
```

Name	Status
controller0	ACTIVE
compute1	ACTIVE
compute0	ACTIVE
ceph0	ACTIVE

- ▶ 2. Log in to **controller0** and **compute0** overcloud nodes to view the unique services running on each node, using the **heat-admin** account provisioned during deployment. The **heat-admin** user on each node is configured with SSH keys for the **stack** user on **director** to allow passwordless access. Use the **docker ps** command to verify the status of the OpenStack services.
- 2.1. Using SSH, log in to the **controller0** overcloud node. List network configuration and verify the status of the OpenStack services, then log out.

```
(undercloud) [stack@director ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ ip addr | grep -E 'eth0|vlan|br-ex'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
    inet 172.25.249.58/24 brd 172.25.249.255 scope global eth0
        inet 172.25.249.50/32 brd 172.25.249.255 scope global eth0
10: br-ex: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.25.250.1/24 brd 172.25.250.255 scope global br-ex
        inet 172.25.250.50/32 brd 172.25.250.255 scope global br-ex
12: vlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.1.1/24 brd 172.24.1.255 scope global vlan10
        inet 172.24.1.51/32 brd 172.24.1.255 scope global vlan10
        inet 172.24.1.50/32 brd 172.24.1.255 scope global vlan10
13: vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.2.1/24 brd 172.24.2.255 scope global vlan20
14: vlan40: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.4.1/24 brd 172.24.4.255 scope global vlan40
        inet 172.24.4.50/32 brd 172.24.4.255 scope global vlan40
15: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.3.1/24 brd 172.24.3.255 scope global vlan30
        inet 172.24.3.50/32 brd 172.24.3.255 scope global vlan30
16: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.5.1/24 brd 172.24.5.255 scope global vlan50
[heat-admin@controller0 ~]$ sudo ovs-vsctl list-ifaces br-ex
eth2
phy-br-ex
[heat-admin@controller0 ~]$ sudo docker ps \
--format 'table {{.Names}}\t{{.Status}}'
NAMES                      STATUS
...output omitted...
neutron_l3_agent           Up 2 hours (healthy)
neutron_metadata_agent     Up 2 hours (healthy)
neutron_dhcp                Up 2 hours (healthy)
...output omitted...
[heat-admin@controller0 ~]$ logout
Connection to controller0 closed.
```

```
(undercloud) [stack@director ~]$
```

- 2.2. Using SSH, log in to the **compute0** node. List the network configuration and verify the status of the OpenStack services, then log out.

```
(undercloud) [stack@director ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ ip addr | grep -E 'eth0|vlan|br-ex'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    group default qlen 1000
        inet 172.25.249.59/24 brd 172.25.249.255 scope global eth0
11: br-ex: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        inet 172.25.250.2/24 brd 172.25.250.255 scope global br-ex
13: vlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        inet 172.24.1.2/24 brd 172.24.1.255 scope global vlan10
14: vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        inet 172.24.2.2/24 brd 172.24.2.255 scope global vlan20
15: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        inet 172.24.3.2/24 brd 172.24.3.255 scope global vlan30
16: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
    group default qlen 1000
        inet 172.24.5.2/24 brd 172.24.5.255 scope global vlan50
[heat-admin@compute0 ~]$ sudo ovs-vsctl list-ifaces br-trunk
eth1
vlan10
vlan20
vlan30
vlan50
[heat-admin@compute0 ~]$ sudo docker ps \
--format 'table {{.Names}}\t{{.Status}}'
NAMES          STATUS
...output omitted...
nova_compute      Up 2 hours (healthy)
...output omitted...
[heat-admin@compute0 ~]$ exit
Connection to compute0 closed.
(undercloud) [stack@director ~]$ exit
Connection to director closed.
[student@workstation ~]$
```

- 3. Use the administrator's environment file to list the services, and endpoints created by the overcloud deployment.

- 3.1. On **workstation**, source the **/home/student/admin-rc** environment file.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$
```

- 3.2. List the OpenStack services in the service catalog. The output shows the name, the type of service, and the endpoints.

```
[student@workstation ~]$ openstack catalog list
```

Name	Type	Endpoints
aodh	alarming	regionOne internal: http://172.24.1.50:8042 regionOne admin: http://172.24.1.50:8042 regionOne public: http://172.25.250.50:8042 regionOne internal: http://172.24.1.50:8977
panko	event	regionOne admin: http://172.24.1.50:8977 regionOne public: http://172.25.250.50:8977 regionOne internal: http://172.24.1.50:8977
<i>...output omitted...</i>		

3.3. Use the **openstack service list** command to list services with the service IDs.

```
[student@workstation ~]$ openstack service list
```

ID	Name	Type
25f5213e9bbd4222874921c40e7dd358	aodh	alarming
27c52b60f4c540aab3068a947ff6b4bf	panko	event
2f17dbd266004ebb9d3aea88f3087feb	placement	placement
436e7d268db14726a2d6af3a244c0a63	heat-cfn	cloudformation
4e8365473eb94d05949ae3012867b0a8	swift	object-store
5e8386b8feb04c50a2885029b184512f	cinderv3	volumev3
5ed96d46365f4bd7a0a82a43ddaeed45	nova	compute
7482a03f42bd4b628286ee1147e323a5	neutron	network
9383173e7b9c42a5b2a2e0ba5044d6cb	glance	image
c4f2867b98444348b03105c434691311	keystone	identity
d26394447bf24012b46d0f7a7a618759	octavia	load-balancer
d8581ea347f240e192227eab04f9a78	cinder	volume
f3b6c9c52879418691a9ceb13abddc5a	gnocchi	metric
f99ea7c2f9eb4e1388f77cea105c7810	cinderv2	volumev2
fd0eb883a100449098088d1a7a8afe44	ceilometer	metering
ff3b69668f484f039d3354430e69c4d0	heat	orchestration

3.4. Use the **openstack endpoint list** command to list the endpoint URLs with the endpoint IDs. Verify that all the endpoints listed are enabled.

```
[student@workstation ~]$ openstack endpoint list \
```

```
-c "ID" -c "Service Name" -c "Enabled" -c "URL"
```

ID	Service Name	Enabled	URL
04ed...0fe1	nova	True	http://172.24.1.50:8774/v2.1
cf57...19a3	nova	True	http://172.25.250.50:8774/v2.1
0dd8...dc61	gnocchi	True	http://172.24.1.50:8041
1295...a377	swift	True	http://172.24.3.50:8080
12c1...5484	heat-cfn	True	http://172.24.1.50:8000/v1

...output omitted...

- 4. Using the administrator's environment file, list the hypervisor nodes and the statistics. Verify the compute services running on these nodes are enabled and in the **up** state.

4.1. List the hypervisor nodes.

```
[student@workstation ~]$ openstack hypervisor list
+-----+-----+-----+-----+
| ID | Hypervisor Hostname           | Hypervisor Type | Host IP   | State |
+-----+-----+-----+-----+
| 1  | compute1.overcloud.example.com | QEMU          | 172.24.1.12 | up    |
| 2  | compute0.overcloud.example.com | QEMU          | 172.24.1.2  | up    |
+-----+-----+-----+-----+
```

4.2. List the hypervisor statistics.

```
[student@workstation ~]$ openstack hypervisor stats show
+-----+-----+
| Field        | Value |
+-----+-----+
| count        | 2      |
| current_workload | 0      |
| disk_available_least | 110    |
| free_disk_gb | 112    |
| free_ram_mb  | 4094   |
| local_gb     | 112    |
| local_gb_used | 0      |
| memory_mb    | 12286  |
| memory_mb_used | 8192   |
| running_vms  | 0      |
| vcpus         | 8      |
| vcpus_used   | 0      |
+-----+-----+
```

4.3. List the compute services running on the overcloud nodes. Verify that the services are enabled and in the **up** state.

```
[student@workstation ~]$ openstack compute service list \
-c "ID" -c "Binary" -c "Host" -c "Status" -c "State"
+-----+-----+-----+-----+
| ID | Binary        | Host          | Status | State |
+-----+-----+-----+-----+
| 1  | nova-scheduler | controller0.overcloud.example.com | enabled | up    |
| 2  | nova-consoleauth | controller0.overcloud.example.com | enabled | up    |
| 3  | nova-conductor  | controller0.overcloud.example.com | enabled | up    |
| 4  | nova-compute    | compute1.overcloud.example.com    | enabled | up    |
| 5  | nova-compute    | compute0.overcloud.example.com    | enabled | up    |
+-----+-----+-----+-----+
```

- 5. Create the identity environment files for the **architect1**, and **developer1** user.

OpenStack project resources

RESOURCE	NAME
Project	finance
Administrative user	<ul style="list-style-type: none"> • Name: architect1 • Project: finance • Role: admin • Password: redhat • Identity environment file: /home/student/architect1-finance-rc
User	<ul style="list-style-type: none"> • Name: developer1 • Project: finance • Role: _member_ • Password: redhat • Identity environment file: /home/student/developer1-finance-rc

- 5.1. Create the **finance** project.

```
[student@workstation ~]$ openstack project create finance
+-----+-----+
| Field      | Value
+-----+-----+
| description | |
| domain_id   | default
| enabled     | True
| id          | 5fe43cde6253439b88f2e16cfcb25ff9
| is_domain   | False
| name        | finance
| parent_id   | default
| tags         | []
+-----+-----+
```

- 5.2. Create the **developer1** and **architect1** users with **redhat** as the password.

```
[student@workstation ~]$ openstack user create \
--project finance \
--password redhat architect1
+-----+-----+
| Field      | Value
+-----+-----+
| default_project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| domain_id   | default
| enabled     | True
| id          | c729dfb7ae4f4391b1f09a9c0a23e57b
| name        | architect1
| options      | {}
| password_expires_at | None
+-----+-----+
[student@workstation ~]$ openstack user create \
```

```
--project finance \
--password redhat developer1
+-----+
| Field          | Value           |
+-----+
| default_project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| domain_id      | default          |
| enabled         | True             |
| id              | 0f4a95f9fe1549bb9cc8ef76d39b2a30 |
| name            | developer1       |
| options          | {}               |
| password_expires_at | None             |
+-----+
```

5.3. Assign the **admin** role to the **architect1** user.

```
[student@workstation ~]$ openstack role add \
--project finance \
--user architect1 admin
[student@workstation ~]$
```

5.4. Assign the **_member_** role to the **developer1** user.

```
[student@workstation ~]$ openstack role add \
--project finance \
--user developer1 _member_
[student@workstation ~]$
```

5.5. Create the identity environment file for the **architect1** user. Save the file as **/home/student/architect1-finance-rc**. The file should read as follows:

```
unset OS_SERVICE_TOKEN
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export OS_PROJECT_NAME="finance"
export OS_PROJECT_DOMAIN_NAME="Default"
export OS_USERNAME="architect1"
export OS_USER_DOMAIN_NAME="Default"
export OS_PASSWORD=redhat
export OS_IDENTITY_API_VERSION=3
export PS1='[\u@\h \w(architect1-finance)]\$ '
```

5.6. Create the identity environment file for the **developer1** user. Save the file as **/home/student/developer1-finance-rc**. The file should read as follows:

```
unset OS_SERVICE_TOKEN
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export OS_PROJECT_NAME="finance"
export OS_PROJECT_DOMAIN_NAME="Default"
export OS_USERNAME="developer1"
export OS_USER_DOMAIN_NAME="Default"
export OS_PASSWORD=redhat
export OS_IDENTITY_API_VERSION=3
export PS1='[\u@\h \w(developer1-finance)]\$ '
```

- 6. As the **architect1** user, import and create the **rhel7-web** image into the image repository according to the following table.

OpenStack image resources

RESOURCE	NAME
Image	<ul style="list-style-type: none">• Name: rhel7• URL: http://materials.example.com/osp-small.qcow2

- 6.1. Source the **/home/student/architect1-finance-rc** environment file.

```
[student@workstation ~]$ source ~/architect1-finance-rc
[student@workstation ~(architect1-finance)]$
```

- 6.2. Download the QCOW2 image from <http://materials.example.com/osp-small.qcow2>.

```
[student@workstation ~(architect1-finance)]$ wget \
http://materials.example.com/osp-small.qcow2
...output omitted...
```

- 6.3. Create an image named **rhel7**.

```
[student@workstation ~(architect1-finance)]$ openstack image create \
--disk-format qcow2 --file osp-small.qcow2 rhel7
...output omitted...
```

- 7. As the **architect1** user, create a flavor named **default** using the following specifications:

- Name: small
- VCPUs: 2
- RAM: 1024 MB
- Root Disk: 10 GB
- Public: true

```
[student@workstation ~(architect1-finance)]$ openstack flavor create \
--vcpus 2 \
--ram 1024 \
--disk 10 \
default
+-----+-----+
| Field          | Value   |
+-----+-----+
| OS-FLV-DISABLED:disabled | False    |
| OS-FLV-EXT-DATA:ephemeral | 0        |
| disk            | 10       |
| id              | 6b9afa45-0d82-40f6-bcf1-8347a81fe0a3 |
| name            | default  |
| os-flavor-access:is_public | True     |
| properties      |          |
| ram             | 1024     |
```

rxtx_factor	1.0	
swap		
vcpus	2	

- 8. As the **architect1** user, create an external network named **provider-datacentre** and its associated subnet, **provider-subnet-172.25.250**. Mark the network as *external*. Create the network resources as listed in the following table:

OpenStack network resource

RESOURCE	NAME
Project network	<ul style="list-style-type: none"> • Name: finance-network1 • Subnet name: finance-subnet1 • Network subnet: 192.168.1.0/24 • DNS server: 172.25.250.254
External network	<ul style="list-style-type: none"> • Name: provider-datacentre • External: Yes • Provider network type: flat • Provider physical network: datacentre • Shared: Yes • Subnet name: provider-subnet-172.25.250 • Network subnet: 172.25.250.0/24 • Network gateway: 172.25.250.254 • Allocation pool: 172.25.250.101,172.25.250.189 • DNS server: 172.25.250.254
Router Name	finance-router1 Set the router's gateway as the external network and define an interface in the private network.

- 8.1. As the **architect1** user, create the **provider-datacentre** network.

```
[student@workstation ~-(architect1-finance)]$ openstack network create \
--external --share \
--provider-network-type flat \
--provider-physical-network datacentre \
provider-datacentre
+-----+-----+
| Field          | Value        |
+-----+-----+
| admin_state_up | UP           |
| availability_zone_hints |          |
| availability_zones |          |
| created_at     | 2018-06-21T15:15:39Z |
| description    |          |
| dns_domain     | None         |
| id             | f4b402ab-fe9f-4b4a-8dc5-495979133315 |
| ipv4_address_scope | None       |
| ipv6_address_scope | None       |
| is_default     | False        |
```

is_vlan_transparent	None
mtu	1500
name	provider-datacentre
port_security_enabled	True
project_id	cbb8de56066b4d40afb9f44bd1cd0af3
provider:network_type	flat
provider:physical_network	datacentre
provider:segmentation_id	None
qos_policy_id	None
revision_number	6
router:external	External
segments	None
shared	True
status	ACTIVE
subnets	
tags	
updated_at	2018-06-21T15:15:40Z

8.2. Create the **provider-subnet-172.25.250** subnet.

[student@workstation ~ (architect1-finance)]\$ openstack subnet create \
--subnet-range 172.25.250.0/24 \
--no-dhcp \
--gateway 172.25.250.254 \
--allocation-pool start=172.25.250.101,end=172.25.250.189 \
--dns-nameserver 172.25.250.254 \
--network provider-datacentre \
provider-subnet-172.25.250
+-----+-----+-----+
Field Value
+-----+-----+-----+
allocation_pools 172.25.250.101-172.25.250.189
cidr 172.25.250.0/24
created_at 2018-06-21T15:17:11Z
description
dns_nameservers 172.25.250.254
enable_dhcp False
gateway_ip 172.25.250.254
host_routes
id 0ddd461c-4e8f-44df-87f1-fa43f755a5a6
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name provider-subnet-172.25.250
network_id f4b402ab-fe9f-4b4a-8dc5-495979133315
project_id cbb8de56066b4d40afb9f44bd1cd0af3
revision_number 0
segment_id None
service_types
subnetpool_id None
tags
updated_at 2018-06-21T15:17:11Z

8.3. Create the **finance-router1** router.

```
[student@workstation ~-(architect1-finance)]$ openstack router create \
finance-router1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T15:45:44Z |
| description | |
| distributed | False |
| external_gateway_info | None |
| flavor_id | None |
| ha | False |
| id | 20515fcf-54c8-4a4a-8c77-c0c3f953cc7a |
| name | finance-router1 |
| project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| revision_number | 1 |
| routes | |
| status | ACTIVE |
| tags | |
| updated_at | 2018-06-21T15:45:44Z |
+-----+-----+
```

8.4. Create the **finance-network1** project network.

```
[student@workstation ~-(architect1-finance)]$ openstack network create \
finance-network1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T15:47:07Z |
| description | |
| dns_domain | None |
| id | 36d73642-f152-4c8e-a2ec-bd12e43c335a |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1450 |
| name | finance-network1 |
| port_security_enabled | True |
| project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| provider:network_type | vxlan |
| provider:physical_network | None |
| provider:segmentation_id | 69 |
| qos_policy_id | None |
| revision_number | 3 |
| router:external | Internal |
+-----+-----+
```

segments	None
shared	False
status	ACTIVE
subnets	
tags	
updated_at	2018-06-21T15:47:07Z

8.5. Create the **finance-subnet1** project subnet.

```
[student@workstation ~-(architect1-finance)]$ openstack subnet create \
--subnet-range 192.168.1.0/24 \
--dhcp \
--dns-nameserver 172.25.250.254 \
--network finance-network1 \
finance-subnet1
+-----+-----+
| Field      | Value   |
+-----+-----+
| allocation_pools | 192.168.1.2-192.168.1.254 |
| cidr       | 192.168.1.0/24 |
| created_at  | 2018-06-21T15:47:48Z |
| description | |
| dns_nameservers | 172.25.250.254 |
| enable_dhcp  | True    |
| gateway_ip  | 192.168.1.1 |
| host_routes | |
| id          | 2112b3c2-58d1-48aa-ab4d-f2ab657c1a15 |
| ip_version   | 4        |
| ipv6_address_mode | None   |
| ipv6_ra_mode  | None   |
| name         | finance-subnet1 |
| network_id   | 36d73642-f152-4c8e-a2ec-bd12e43c335a |
| project_id   | 5fe43cde6253439b88f2e16cfcb25ff9 |
| revision_number | 0      |
| segment_id   | None   |
| service_types | |
| subnetpool_id | None   |
| tags          | |
| updated_at   | 2018-06-21T15:47:48Z |
+-----+-----+
```

8.6. Connect the router to the project network, **finance-subnet1**.

```
[student@workstation ~-(architect1-finance)]$ openstack router add \
subnet finance-router1 finance-subnet1
[student@workstation ~-(architect1-finance)]$
```

8.7. Set the router as the external gateway for the external network, **provider-datacentre**.

```
[student@workstation ~-(architect1-finance)]$ openstack router set \
--external-gateway provider-datacentre finance-router1
[student@workstation ~-(architect1-finance)]$
```

- 9. As the **developer1** user, create the security rule in the default security group, **default** to allow the SSH port. Create the **example-keypair** key pair using the public key stored in <http://materials.example.com/keypairs/example-keypair.pub>. The following table lists the details about the resources.

OpenStack security resources

RESOURCE	NAME
Security group rule	<ul style="list-style-type: none"> • Security group: default • Protocol: TCP • Port: 22
Key pair	<ul style="list-style-type: none"> • Name: example-keypair • Public key: example-keypair.pub

- 9.1. Source the **developer1** identity environment file.

```
[student@workstation ~-(architect1-finance)]$ source ~/developer1-finance-rc
[student@workstation ~-(developer1-finance)]$
```

- 9.2. Add a security group rule in the **default** security group to allow access using the port 22.

```
[student@workstation ~-(developer1-finance)]$ openstack security group rule \
create --protocol tcp --dst-port 22 \
default
+-----+-----+
| Field      | Value           |
+-----+-----+
| created_at | 2018-06-21T15:49:52Z |
| description |                 |
| direction   | ingress          |
| ether_type  | IPv4             |
| id          | 5735dd6a-629c-411e-9331-87941eab01f5 |
| name        | None             |
| port_range_max | 22               |
| port_range_min | 22               |
| project_id  | 5fe43cde6253439b88f2e16cfcb25ff9 |
| protocol    | tcp              |
| remote_group_id | None            |
| remote_ip_prefix | 0.0.0.0/0       |
| revision_number | 0                |
| security_group_id | b00d9c6f-8c5b-4f10-888b-13d1417079cf |
| updated_at   | 2018-06-21T15:49:52Z |
+-----+-----+
```

- 9.3. Download the public key from <http://materials.example.com/keypairs/example-keypair.pub>. Create the key pair **example-keypair** and using the **example-keypair.pub** public key.

```
[student@workstation ~-(developer1-finance)]$ wget \
http://materials.example.com/keypairs/example-keypair.pub
...output omitted...
[student@workstation ~-(developer1-finance)]$ openstack keypair create \
```

```
--public-key example-keypair.pub example-keypair
+-----+
| Field      | Value
+-----+
| fingerprint | 31:b5:bb:2c:a0:e7:39:5d:5f:5e:0e:20:8b:a5:c2:1f |
| name       | example-keypair
| user_id    | 0f4a95f9fe1549bb9cc8ef76d39b2a30
+-----+
```

- 10. Launch an instance in the environment according to the following table.

OpenStack instance resources

RESOURCE	NAME
Instance name	finance-server1
Image name	rhel7
Flavor name	default
Key pair	example-keypair
Network name	finance-network1

- 10.1. Use the **developer1** environment's file to launch an instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--key-name example-keypair \
--nic net-id=finance-network1 \
--wait finance-server1
...output omitted...
```

- 10.2. Review the status of the instance to ensure it is marked as **ACTIVE**.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server1",
    "Image": "rhel7-web",
    "ID": "efb26df0-6fe3-490c-89ef-4889ed9f975c",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.1"
  }
]
```

- 11. As the **operator1** user, create a floating IP. Associate the floating IP to the **finance-server1** instance.

- 11.1. From the floating IPs pool **provider-datacentre**, create a floating IP.

```
[student@workstation ~ (developer1-finance)]$ openstack floating ip create \
```

provider-datacentre	
Field	Value
created_at	2018-06-21T15:56:36Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.P
floating_network_id	f4b402ab-fe9f-4b4a-8dc5-495979133315
id	7404fea7-566b-4bd8-a636-d33319521f8b
name	172.25.250.P
port_id	None
project_id	5fe43cde6253439b88f2e16cfcb25ff9
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-21T15:56:36Z

11.2. Associate the floating IP to the instance.

```
[student@workstation ~ (developer1-finance)]$ openstack server add \
floating ip finance-server1 172.25.250.P
```

11.3. Ensure that the IP address has been successfully assigned.

```
[student@workstation ~ (developer1-finance)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "finance-server1",
    "Image": "rhel7-web",
    "ID": "efb26df0-6fe3-490c-89ef-4889ed9f975c",
    "Flavor": "default",
    "Networks": "finance-network1=192.168.1.N, 172.25.250.P"
  }
]
```

► 12. Use SSH to connect to the instance and exit from the instance.



NOTE

Allow a few minutes for the **cloud-init** service to complete.

```
[student@workstation ~ (developer1-finance)]$ ssh cloud-user@172.25.250.P
[cloud-user@finance-server1 ~]$ exit
[student@workstation ~ (developer1-finance)]$
```

Cleanup

From **workstation**, run the **lab verify-overcloud** script with the **cleanup** argument to clean up this exercise.

```
[student@workstation ~]$ lab verify-overcloud cleanup
```

This concludes the guided exercise.

DEPLOYING AN OPENSTACK OVERCLOUD

PERFORMANCE CHECKLIST

In this lab, you will import an overcloud plan into the Red Hat OpenStack Platform director web UI and initiate deployment using one controller and one compute node. You will ensure that the OpenStack services are running. You will launch an instance in the environment and check connectivity by connecting to the instance using SSH.

OUTCOMES

You should be able to:

- Create an overcloud plan directory with the director's base templates and include the plan environment file based on a particular deployment architecture.
- Import an overcloud plan with single NIC VLANs for isolating internal overcloud networks using the web UI.
- Deploy OpenStack Overcloud using the director web UI.
- Review the system configuration after deployment of the OpenStack overcloud.
- Verify OpenStack overcloud deployment.
- Create a base environment comprised of an image, a network environment, and a key pair.
- Launch an instance in the new overcloud deployment.

Confirm that the **workstation** and the overcloud's virtual machines are started.

Log in to workstation as **student** using **student** as the password. Run the **lab deploy-overcloud-lab setup** command, which performs the following tasks:

- Removes the **overcloud** deployment.
- Ensures that the **controller0** and **compute0** bare metal nodes are configured.
- Deletes any existing overcloud plans.

```
[student@workstation ~]$ lab deploy-overcloud-lab setup
```

1. Copy **/usr/share/openstack-tripleo-heat-templates** from the **director** as the **stack** user to **/home/student/overcloud-poc** on **workstation**.
2. Download and extract <http://materials.example.com/classroom-plan.tar.gz> in **/home/student/overcloud-poc**. The **classroom-plan.tar.gz** contains the **plan-environment.yaml** and environment files defining overcloud network isolation using single NIC VLANs. The overcloud plan is customized to provide the following features:
 - Use single NIC VLANs for isolating overcloud networks.
 - External access to overcloud deployment using **eth2** on the controller virtual machine.
 - Use fixed virtual IP addresses for public endpoints of OpenStack service.
 - Use local container repository that exists on **director**.
 - Disable high availability of controller node.

- Disable telemetry.
3. Use the `https://director-ui.lab.example.com` to access the Director web UI. In the initial login screen, use **admin** as user name and **redhat** as the password to access the web UI.
 4. In the Red Hat OpenStack Platform director web UI, import an overcloud plan by uploading the files stored in the `/home/student/overcloud-poc`. Enter the overcloud plan name as **overcloud-poc**.
 5. Ensure that one **Controller** role and one **Compute** role are assigned to the bare metal nodes.
 6. Once the overcloud plan is imported and configured, start the overcloud deployment.



NOTE

If all the pre-deployment validations do not pass, a warning message appears. For the classroom deployment, this is expected.

7. Verify that the **overcloud-poc** stack shows **CREATE_COMPLETE** status and overcloud nodes are in **ACTIVE** state.
8. Use the OpenStack dashboard to download the **admin** identity environment file. Modify the environment file to use the admin's password listed after the overcloud deployment in the director web UI.
9. Source the identity environment file located at `/home/student/Downloads/admin-openrc.sh` and create the resources indicated in the following table. Create the identity environment files for the **architect1** and **operator1** users.

OpenStack project resources

RESOURCE	NAME
Project	production
Administrative user	<ul style="list-style-type: none"> • Name: architect1 • Project: production • Role: admin • Password: redhat • Identity environment file: <code>/home/student/architect1-production-rc</code>
User	<ul style="list-style-type: none"> • Name: operator1 • Project: production • Role: _member_ • Password: redhat • Identity environment file: <code>/home/student/operator1-production-rc</code>

10. As the **architect1** user, import and create the **rhe17-web** image into the image repository according to the following table.

OpenStack image resources

RESOURCE	NAME
Image	<ul style="list-style-type: none">Name: rhe17-webURL: http://materials.example.com/osp-web.qcow2

11. As the **architect1** user, create a **default** flavor using the following specifications:
- Name: small
 - VCPUs: 2
 - RAM: 1024 MB
 - Root Disk: 10 GB
 - Public: true
12. As the **architect1** user, create an external network named **provider-datacentre** and its associated subnet, **provider-subnet-172.25.250**. Mark the network as *external*. Create the network resources as listed in the following table:

OpenStack network resource

RESOURCE	NAME
Private Network	<ul style="list-style-type: none">Name: production-network1Subnet name: production-subnet1Network subnet: 192.168.1.0/24DNS server: 172.25.250.254
External Network	<ul style="list-style-type: none">Name: provider-datacentreExternal: YesProvider network type: flatProvider physical network: datacentreShared: YesSubnet name: provider-subnet-172.25.250Network subnet: 172.25.250.0/24Network gateway: 172.25.250.254Allocation pool: 172.25.250.101,172.25.250.189DNS server: 172.25.250.254
Router Name	production-router1 Define the router as the gateway for the external network and define an interface in the private network.

13. As the **operator1** user, create the security rule in the default security group, **default** to allow the SSH port. Create the **example-keypair** key pair using the public key stored in

<http://materials.example.com/keypairs/example-keypair.pub>. The following table lists the details about the resources.

OpenStack security resources

RESOURCE	NAME
Security Group Rule	<ul style="list-style-type: none">Security group: defaultProtocol: TCPPort: 22
Key pair	<ul style="list-style-type: none">Name: example-keypairPublic key: example-keypair.pub

- As the **operator1** user, launch the instance **production-server1** in the environment using the following resources:

OpenStack instance resources

RESOURCE	NAME
Name	production-server1
Image	rhel7-web
Flavor	default
Key pair	example-keypair
Network name	production-network1

- As the **developer1** user, create a floating IP. Associate the floating IP to the **production-server1**.
- Use SSH to connect to the instance and exit from the instance.



NOTE

Allow a few minutes for the **cloud-init** service to complete.

Evaluation

From **workstation**, run the **lab deploy-overcloud-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab deploy-overcloud-lab grade
```

Cleanup

From **workstation**, run the **lab deploy-overcloud-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab deploy-overcloud-lab cleanup
```

This concludes the lab.

DEPLOYING AN OPENSTACK OVERCLOUD

PERFORMANCE CHECKLIST

In this lab, you will import an overcloud plan into the Red Hat OpenStack Platform director web UI and initiate deployment using one controller and one compute node. You will ensure that the OpenStack services are running. You will launch an instance in the environment and check connectivity by connecting to the instance using SSH.

OUTCOMES

You should be able to:

- Create an overcloud plan directory with the director's base templates and include the plan environment file based on a particular deployment architecture.
- Import an overcloud plan with single NIC VLANs for isolating internal overcloud networks using the web UI.
- Deploy OpenStack Overcloud using the director web UI.
- Review the system configuration after deployment of the OpenStack overcloud.
- Verify OpenStack overcloud deployment.
- Create a base environment comprised of an image, a network environment, and a key pair.
- Launch an instance in the new overcloud deployment.

Confirm that the **workstation** and the overcloud's virtual machines are started.

Log in to workstation as **student** using **student** as the password. Run the **lab deploy-overcloud-lab setup** command, which performs the following tasks:

- Removes the **overcloud** deployment.
- Ensures that the **controller0** and **compute0** bare metal nodes are configured.
- Deletes any existing overcloud plans.

```
[student@workstation ~]$ lab deploy-overcloud-lab setup
```

1. Copy **/usr/share/openstack-tripleo-heat-templates** from the **director** as the **stack** user to **/home/student/overcloud-poc** on **workstation**.

```
[student@workstation ~]$ scp -r \
stack@director:/usr/share/openstack-tripleo-heat-templates \
~/overcloud-poc
...output omitted...
```

2. Download and extract <http://materials.example.com/classroom-plan.tar.gz> in **/home/student/overcloud-poc**. The **classroom-plan.tar.gz** contains the **plan-environment.yaml** and environment files defining overcloud network isolation using single NIC VLANs. The overcloud plan is customized to provide the following features:

- Use single NIC VLANs for isolating overcloud networks.
- External access to overcloud deployment using **eth2** on the controller virtual machine.
- Use fixed virtual IP addresses for public endpoints of OpenStack service.
- Use local container repository that exists on **director**.
- Disable high availability of controller node.
- Disable telemetry.

2.1. Review the **plan-environment.yaml** in **/home/student/overcloud-poc**.

```
[student@workstation ~]$ cd overcloud-poc
[student@workstation overcloud-poc]$ cat plan-environment.yaml
version: 1.0

name: overcloud
description: >
    Default Deployment plan
template: overcloud.yaml
environments:
    - path: overcloud-resource-registry-puppet.yaml
    - path: environments/docker.yaml
    - path: environments/docker-ha.yaml
```

2.2. Download <http://materials.example.com/classroom-plan.tar.gz> in **/home/student/overcloud-poc**.

```
[student@workstation overcloud-poc]$ wget \
http://materials.example.com/classroom-plan.tar.gz
...output omitted...
```

2.3. Extract **classroom-plan.tar.gz**.

```
[student@workstation overcloud-poc]$ tar xzvf \
classroom-plan.tar.gz
...output omitted...
```

2.4. Review the **plan-environment.yaml** downloaded and extracted in **/home/student/overcloud-poc**.

```
[student@workstation overcloud-poc]$ cat plan-environment.yaml
description: 'Default Deployment plan

'

environments:
    - path: overcloud-resource-registry-puppet.yaml
    - path: environments/docker.yaml
    - path: classroom-environments/net-single-nic-with-vlans.yaml
    - path: environments/network-isolation.yaml
    - path: environments/fixed-ip-vips.yaml
    - path: environments/disable-telemetry.yaml
name: overcloud-poc
parameter_defaults:
...output omitted...
```

3. Use the `https://director-ui.lab.example.com` to access the Director web UI. In the initial login screen, use **admin** as user name and **redhat** as the password to access the web UI.

 - 3.1. Open the `https://director-ui.lab.example.com` to access the Director web UI. If a certificate error appears, accept the self-signed certificate.
 - 3.2. Log in to the director web UI using **admin** as user name and **redhat** as the password.
4. In the Red Hat OpenStack Platform director web UI, import an overcloud plan by uploading the files stored in the `/home/student/overcloud-poc`. Enter the overcloud plan name as **overcloud-poc**.

 - 4.1. From the Red Hat OpenStack Platform Director dashboard, click Import Plan in the Plans tab.
 - 4.2. In the New Plan tab, enter **overcloud-poc** as the Plan Name.
 - 4.3. Select the Upload Type as Local Folder.
 - 4.4. Click Browse to select the Plan Files. Browse and select **overcloud-poc** directory in the **student** home directory. Click Upload.
 - 4.5. Click Upload Files and Create Plan. Wait till the overcloud plan files are uploaded and the plan is created on the **director** node. When complete a notification box appears on the dashboard.
5. Ensure that one **Controller** role and one **Compute** role are assigned to the bare metal nodes.

 - 5.1. From the Plans tab, click the **overcloud-poc** plan card. The web page shows four steps to deploy the **overcloud-poc** plan.
 - 5.2. In the Configure Roles and Assign Nodes section, ensure that two nodes are selected. One of the nodes is assigned the **Controller** role and other is assigned the **Compute** role.
6. Once the overcloud plan is imported and configured, start the overcloud deployment.



NOTE

If all the pre-deployment validations do not pass, a warning message appears. For the classroom deployment, this is expected.

- 6.1. Click Validate and Deploy to start the deployment validation.



NOTE

If all the pre-deployment validations do not pass, a warning message appears. For the classroom deployment, this is expected.

- 6.2. Click Deploy to start the overcloud deployment.
- 6.3. The Director dashboard monitors the progress of the overcloud deployment and displays a progress bar that indicates the overall progress. In case, the Plan **overcloud-**

poc deployment screen is closed, use the **View detailed information** link to list or filter resources created by the deployment of orchestration stack.

Enter **PROGRESS** in the filter text box to filter resources those are in the **CREATE_IN_PROGRESS** state.

- 6.4. After the overcloud deployment process completes the Deploy section displays the Overcloud IP address, Username, and Password to access the overcloud dashboard. The administrator's password in your case might differ.

Overcloud information:

```
Overcloud IP address: 172.25.250.50
Username: admin
Password: NQH329y7PHcyM6HPTu2MHUBvM
```

7. Verify that the **overcloud-poc** stack shows **CREATE_COMPLETE** status and overcloud nodes are in **ACTIVE** state.

- 7.1. Using SSH, log in to the **director**.

```
[student@workstation overcloud-poc]$ ssh stack@director
(undercloud) [stack@director ~]$
```

- 7.2. Use **openstack stack list** to verify the status of the **overcloud-poc** stack.

```
(undercloud) [stack@director ~]$ openstack stack list \
-c "Stack Name" -c "Stack Status"
+-----+-----+
| Stack Name      | Stack Status    |
+-----+-----+
| overcloud-poc   | CREATE_COMPLETE |
+-----+-----+
```

- 7.3. Use **openstack server list** to verify the status of overcloud nodes.

```
(undercloud) [stack@director ~]$ openstack server list \
-c "Name" -c "Status"
+-----+-----+
| Name            | Status |
+-----+-----+
| overcloud-poc-compute-0 | ACTIVE |
| overcloud-poc-controller-0 | ACTIVE |
+-----+-----+
```

Log out of **director**.

```
(undercloud) [stack@director ~]$ logout
Connection to director closed.
[student@workstation overcloud-poc]$
```

8. Use the OpenStack dashboard to download the **admin** identity environment file. Modify the environment file to use the admin's password listed after the overcloud deployment in the director web UI.
 - 8.1. From a new tab in the browser, log in to `http://dashboard.overcloud.example.com` using **admin** and the password listed after the overcloud deployment in the director web UI.
 - 8.2. Click the Project tab at the top of the page. Navigate to Project → API Access.
 - 8.3. On the API Access page, navigate to Download OpenStack RC File → OpenStack RC File (Identity API v3). Save the environment file.
 - Log out from the dashboard.
 - 8.4. Edit the `/home/student/Downloads/admin-openrc.sh` file to use the administrator's password retrieved from the director web UI. The text highlighted in the screen output represents text that has been edited. The administrator's password in your case might differ.

```
[student@workstation overcloud-poc]$ cd
[student@workstation ~]$ vi ~/Downloads/admin-openrc.sh
...output omitted...
export OS_USERNAME="admin"
# With Keystone you pass the keystone password.
# echo "Please enter your OpenStack Password for project $OS_PROJECT_NAME as user
$OS_USERNAME: "
# read -sr OS_PASSWORD_INPUT
export OS_PASSWORD=NQH329y7PHcyM6HPTu2MHUBvM
...output omitted...
```

Save the changes to the file.

- 8.5. Log out from the director web UI.
9. Source the identity environment file located at `/home/student/Downloads/admin-openrc.sh` and create the resources indicated in the following table. Create the identity environment files for the **architect1** and **operator1** users.

OpenStack project resources

RESOURCE	NAME
Project	production
Administrative user	<ul style="list-style-type: none"> • Name: architect1 • Project: production • Role: admin • Password: redhat • Identity environment file: /home/student/architect1-production-rc

RESOURCE	NAME
User	<ul style="list-style-type: none"> Name: operator1 Project: production Role: _member_ Password: redhat Identity environment file: /home/student/operator1-production-rc

- 9.1. Source the **admin** environment file in **/home/student/Downloads/admin-openrc.sh**.

```
[student@workstation ~]$ source ~/Downloads/admin-openrc.sh
[student@workstation ~]$
```

- 9.2. Create the **production** project.

```
[student@workstation ~]$ openstack project create production
+-----+-----+
| Field      | Value   |
+-----+-----+
| description |          |
| domain_id  | default  |
| enabled     | True    |
| id          | 5fe43cde6253439b88f2e16cfcb25ff9 |
| is_domain   | False   |
| name        | production |
| parent_id   | default  |
| tags         | []      |
+-----+-----+
```

- 9.3. Create the **operator1** and **architect1** users with **redhat** as the password.

```
[student@workstation ~]$ openstack user create \
--project production \
--password redhat architect1
+-----+-----+
| Field      | Value   |
+-----+-----+
| default_project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| domain_id  | default  |
| enabled     | True    |
| id          | c729dfb7ae4f4391b1f09a9c0a23e57b |
| name        | architect1 |
| options     | {}      |
| password_expires_at | None   |
+-----+-----+
[student@workstation ~]$ openstack user create \
--project production \
--password redhat operator1
+-----+-----+
| Field      | Value   |
+-----+-----+
| default_project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| domain_id  | default  |
```

```
| enabled           | True          |
| id               | 0f4a95f9fe1549bb9cc8ef76d39b2a30 |
| name             | operator1    |
| options          | {}            |
| password_expires_at | None          |
+-----+-----+
```

9.4. Assign the **admin** role to the **architect1** user.

```
[student@workstation ~]$ openstack role add \
--project production \
--user architect1 admin
[student@workstation ~]$
```

9.5. Assign the **_member_** role to the **operator1** user.

```
[student@workstation ~]$ openstack role add \
--project production \
--user operator1 _member_
[student@workstation ~]$
```

9.6. Create the identity environment file for the **architect1** user. Save the file as **/home/student/architect1-production-rc**. The file should read as follows:

```
unset OS_SERVICE_TOKEN
unset OS_PROJECT_ID
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export OS_PROJECT_NAME="production"
export OS_PROJECT_DOMAIN_NAME="Default"
export OS_USERNAME="architect1"
export OS_USER_DOMAIN_NAME="Default"
export OS_PASSWORD=redhat
export OS_IDENTITY_API_VERSION=3
export PS1='[\u@\h \w(architect1-production)]\$ '
```

9.7. Create the identity environment file for the **operator1** user. Save the file as **/home/student/operator1-production-rc**. The file should read as follows:

```
unset OS_SERVICE_TOKEN
unset OS_PROJECT_ID
export OS_AUTH_URL=http://172.25.250.50:5000/v3
export OS_PROJECT_NAME="production"
export OS_PROJECT_DOMAIN_NAME="Default"
export OS_USERNAME="operator1"
export OS_USER_DOMAIN_NAME="Default"
export OS_PASSWORD=redhat
export OS_IDENTITY_API_VERSION=3
export PS1='[\u@\h \w(operator1-production)]\$ '
```

10. As the **architect1** user, import and create the **rhel7-web** image into the image repository according to the following table.

OpenStack image resources

RESOURCE	NAME
Image	<ul style="list-style-type: none">Name: rhel7-webURL: http://materials.example.com/osp-web.qcow2

- 10.1. Source the **/home/student/architect1-production-rc** environment file.

```
[student@workstation ~]$ source ~/architect1-production-rc  
[student@workstation ~(architect1-production)]$
```

- 10.2. Download the QCOW2 image from <http://materials.example.com/osp-web.qcow2>.

```
[student@workstation ~(architect1-production)]$ wget \  
http://materials.example.com/osp-web.qcow2  
...output omitted...
```

- 10.3. Create an image named **rhel7-web**.

```
[student@workstation ~(architect1-production)]$ openstack image create \  
--disk-format qcow2 --file osp-web.qcow2 rhel7-web  
...output omitted...
```

11. As the **architect1** user, create a **default** flavor using the following specifications:

- Name: small
- VCPUs: 2
- RAM: 1024 MB
- Root Disk: 10 GB
- Public: true

```
[student@workstation ~(architect1-production)]$ openstack flavor create \  
--vcpus 2 \  
--ram 1024 \  
--disk 10 \  
default  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| OS-FLV-DISABLED:disabled | False |  
| OS-FLV-EXT-DATA:ephemeral | 0 |  
| disk | 10 |  
| id | 6b9afa45-0d82-40f6-bcf1-8347a81fe0a3 |  
| name | default |  
| os-flavor-access:is_public | True |  
| properties | |  
| ram | 1024 |  
| rxtx_factor | 1.0 |  
| swap | |
```

vcpus	2	
+-----+-----+	+-----+	+-----+

12. As the **architect1** user, create an external network named **provider-datacentre** and its associated subnet, **provider-subnet-172.25.250**. Mark the network as *external*. Create the network resources as listed in the following table:

OpenStack network resource

RESOURCE	NAME
Private Network	<ul style="list-style-type: none"> • Name: production-network1 • Subnet name: production-subnet1 • Network subnet: 192.168.1.0/24 • DNS server: 172.25.250.254
External Network	<ul style="list-style-type: none"> • Name: provider-datacentre • External: Yes • Provider network type: flat • Provider physical network: datacentre • Shared: Yes • Subnet name: provider-subnet-172.25.250 • Network subnet: 172.25.250.0/24 • Network gateway: 172.25.250.254 • Allocation pool: 172.25.250.101,172.25.250.189 • DNS server: 172.25.250.254
Router Name	production-router1 Define the router as the gateway for the external network and define an interface in the private network.

- 12.1. As the **architect1** user, create the **provider-datacentre** network.

```
[student@workstation ~ (architect1-production)]$ openstack network create \
--external --share \
--provider-network-type flat \
--provider-physical-network datacentre \
provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T15:15:39Z |
| description | |
| dns_domain | None |
| id | f4b402ab-fe9f-4b4a-8dc5-495979133315 |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1500 |
| name | provider-datacentre |
| port_security_enabled | True |
```

project_id	cbb8de56066b4d40afb9f44bd1cd0af3
provider:network_type	flat
provider:physical_network	datacentre
provider:segmentation_id	None
qos_policy_id	None
revision_number	6
router:external	External
segments	None
shared	True
status	ACTIVE
subnets	
tags	
updated_at	2018-06-21T15:15:40Z

12.2. Create the **provider-subnet-172.25.250** subnet.

[student@workstation ~-(architect1-production)]\$ openstack subnet create \
--subnet-range 172.25.250.0/24 \
--no-dhcp \
--gateway 172.25.250.254 \
--allocation-pool start=172.25.250.101,end=172.25.250.189 \
--dns-nameserver 172.25.250.254 \
--network provider-datacentre \
provider-subnet-172.25.250
+-----+-----+
Field Value
+-----+-----+
allocation_pools 172.25.250.101-172.25.250.189
cidr 172.25.250.0/24
created_at 2018-06-21T15:17:11Z
description
dns_nameservers 172.25.250.254
enable_dhcp False
gateway_ip 172.25.250.254
host_routes
id 0ddd461c-4e8f-44df-87f1-fa43f755a5a6
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name provider-subnet-172.25.250
network_id f4b402ab-fe9f-4b4a-8dc5-495979133315
project_id cbb8de56066b4d40afb9f44bd1cd0af3
revision_number 0
segment_id None
service_types
subnetpool_id None
tags
updated_at 2018-06-21T15:17:11Z
+-----+-----+

12.3. Create the **production-router1** router.

```
[student@workstation ~-(architect1-production)]$ openstack router create \
```

```
production-router1
```

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-21T15:45:44Z
description	
distributed	False
external_gateway_info	None
flavor_id	None
ha	False
id	20515fcf-54c8-4a4a-8c77-c0c3f953cc7a
name	production-router1
project_id	5fe43cde6253439b88f2e16cfcb25ff9
revision_number	1
routes	
status	ACTIVE
tags	
updated_at	2018-06-21T15:45:44Z

12.4. Create the **production-network1** project network.

```
[student@workstation ~ (architect1-production)]$ openstack network create \
production-network1
```

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-21T15:47:07Z
description	
dns_domain	None
id	36d73642-f152-4c8e-a2ec-bd12e43c335a
ipv4_address_scope	None
ipv6_address_scope	None
is_default	False
is_vlan_transparent	None
mtu	1450
name	production-network1
port_security_enabled	True
project_id	5fe43cde6253439b88f2e16cfcb25ff9
provider:network_type	vxlan
provider:physical_network	None
provider:segmentation_id	69
qos_policy_id	None
revision_number	3
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	
tags	
updated_at	2018-06-21T15:47:07Z

12.5. Create the **production-subnet1** project subnet.

```
[student@workstation ~-(architect1-production)]$ openstack subnet create \
--subnet-range 192.168.1.0/24 \
--dhcp \
--dns-nameserver 172.25.250.254 \
--network production-network1 \
production-subnet1
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 192.168.1.2-192.168.1.254 |
| cidr | 192.168.1.0/24 |
| created_at | 2018-06-21T15:47:48Z |
| description | |
| dns_nameservers | 172.25.250.254 |
| enable_dhcp | True |
| gateway_ip | 192.168.1.1 |
| host_routes | |
| id | 2112b3c2-58d1-48aa-ab4d-f2ab657c1a15 |
| ip_version | 4 |
| ipv6_address_mode | None |
| ipv6_ra_mode | None |
| name | production-subnet1 |
| network_id | 36d73642-f152-4c8e-a2ec-bd12e43c335a |
| project_id | 5fe43cde6253439b88f2e16cfcb25ff9 |
| revision_number | 0 |
| segment_id | None |
| service_types | |
| subnetpool_id | None |
| tags | |
| updated_at | 2018-06-21T15:47:48Z |
+-----+-----+
```

12.6. Connect the router to the project network, **production-subnet1**.

```
[student@workstation ~-(architect1-production)]$ openstack router add \
subnet production-router1 production-subnet1
[student@workstation ~-(architect1-production)]$
```

12.7. Set the router as the external gateway for the external network, **provider-datacentre**.

```
[student@workstation ~-(architect1-production)]$ openstack router set \
--external-gateway provider-datacentre production-router1
[student@workstation ~-(architect1-production)]$
```

13. As the **operator1** user, create the security rule in the default security group, **default** to allow the SSH port. Create the **example-keypair** key pair using the public key stored in

<http://materials.example.com/keypairs/example-keypair.pub>. The following table lists the details about the resources.

OpenStack security resources

RESOURCE	NAME
Security Group Rule	<ul style="list-style-type: none">• Security group: default• Protocol: TCP• Port: 22
Key pair	<ul style="list-style-type: none">• Name: example-keypair• Public key: example-keypair.pub

- 13.1. Source the **operator1** identity environment file.

```
[student@workstation ~-(architect1-production)]$ source ~/operator1-production-rc  
[student@workstation ~-(operator1-production)]$
```

- 13.2. Add a security group rule in the **default** security group to allow access using the port 22.

```
[student@workstation ~-(operator1-production)]$ openstack security group rule \  
create --protocol tcp --dst-port 22 \  
default  
+-----+-----+  
| Field      | Value          |  
+-----+-----+  
| created_at | 2018-06-21T15:49:52Z |  
| description |                |  
| direction   | ingress        |  
| ether_type  | IPv4          |  
| id          | 5735dd6a-629c-411e-9331-87941eab01f5 |  
| name        | None          |  
| port_range_max | 22           |  
| port_range_min | 22           |  
| project_id  | 5fe43cde6253439b88f2e16cfcb25ff9 |  
| protocol    | tcp           |  
| remote_group_id | None         |  
| remote_ip_prefix | 0.0.0.0/0     |  
| revision_number | 0            |  
| security_group_id | b00d9c6f-8c5b-4f10-888b-13d1417079cf |  
| updated_at   | 2018-06-21T15:49:52Z |  
+-----+-----+
```

- 13.3. Download the public key from <http://materials.example.com/keypairs/example-keypair.pub>. Create the **example-keypair** key pair using the **example-keypair.pub** public key.

```
[student@workstation ~-(operator1-production)]$ wget \  
http://materials.example.com/keypairs/example-keypair.pub  
...output omitted...  
[student@workstation ~-(operator1-production)]$ openstack keypair create \  
--public-key example-keypair.pub example-keypair  
+-----+
```

Field	Value
fingerprint	31:b5:bb:2c:a0:e7:39:5d:5f:5e:0e:20:8b:a5:c2:1f
name	example-keypair
user_id	0f4a95f9fe1549bb9cc8ef76d39b2a30

14. As the **operator1** user, launch the instance **production-server1** in the environment using the following resources:

OpenStack instance resources

RESOURCE	NAME
Name	production-server1
Image	rhel7-web
Flavor	default
Key pair	example-keypair
Network name	production-network1

- 14.1. Launch an instance named **production-server1** in the environment.

```
[student@workstation ~ (operator1-production)]$ openstack server create \
--image rhel7-web \
--flavor default \
--key-name example-keypair \
--nic net-id=production-network1 \
--wait production-server1
...output omitted...
```

- 14.2. Review the status of the instance to ensure that it is marked as **ACTIVE**.

```
[student@workstation ~ (operator1-production)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "production-server1",
    "Image": "rhel7-web",
    "ID": "efb26df0-6fe3-490c-89ef-4889ed9f975c",
    "Flavor": "default",
    "Networks": "production-network1=192.168.1.N"
  }
]
```

15. As the **developer1** user, create a floating IP. Associate the floating IP to the **production-server1**.

- 15.1. From the floating IPs pool **provider-datacentre**, create the floating IP **172.25.250.P**.

```
[student@workstation ~ (operator1-production)]$ openstack floating ip create \
provider-datacentre
```

Field	Value
created_at	2018-06-21T15:56:36Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.P
floating_network_id	f4b402ab-fe9f-4b4a-8dc5-495979133315
id	7404fea7-566b-4bd8-a636-d33319521f8b
name	172.25.250.P
port_id	None
project_id	5fe43cde6253439b88f2e16cfcb25ff9
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-21T15:56:36Z

15.2. Associate the floating IP to the instance.

```
[student@workstation ~ (operator1-production)]$ openstack server add \
floating ip production-server1 172.25.250.P
```

15.3. Ensure the IP address has been successfully assigned.

```
[student@workstation ~ (operator1-production)]$ openstack server list -f json
[
  {
    "Status": "ACTIVE",
    "Name": "production-server1",
    "Image": "rhel7-web",
    "ID": "efb26df0-6fe3-490c-89ef-4889ed9f975c",
    "Flavor": "default",
    "Networks": "production-network1=192.168.1.N, 172.25.250.P"
  }
]
```

16. Use SSH to connect to the instance and exit from the instance.



NOTE

Allow a few minutes for the **cloud-init** service to complete.

```
[student@workstation ~ (operator1-production)]$ ssh cloud-user@172.25.250.P
[cloud-user@production-server1 ~]$ exit
[student@workstation ~ (operator1-production)]$
```

Evaluation

From **workstation**, run the **lab deploy-overcloud-lab grade** command to confirm success of this exercise.

```
[student@workstation ~]$ lab deploy-overcloud-lab grade
```

Cleanup

From **workstation**, run the **lab deploy-overcloud-lab cleanup** script to clean up this exercise.

```
[student@workstation ~]$ lab deploy-overcloud-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Red Hat OpenStack Platform Director is a tool used to install and manage the deployment and life cycle of Red Hat OpenStack Platform. The director uses OpenStack components running on the undercloud to install an operational OpenStack cloud (the overcloud).
- Director can perform environment health checks, auto-scale an overcloud by adding or replacing nodes, manage minor release updates and major version upgrades, plus patching, monitoring, and regulation compliance.
- An enterprise production cloud is known as an overcloud. Undercloud and overcloud utilize the same technologies, but manage different workloads. Undercloud manages cloud infrastructure, while overcloud manages production and tenant workloads.
- Red Hat OpenStack Platform director web UI provides users with a web-based tool to deploy an overcloud.
- An overcloud deployment plan consists of the orchestration templates and the environment files used to deploy an overcloud. The Director core orchestration heat templates are stored in **/usr/share/openstack-tripleo-heat-templates** directory. The default templates cover a majority of common use cases and designs.
- The **heat-admin** user is configured by the orchestration to access and configure the overcloud nodes using SSH. From **director**, the **stack** user has passwordless SSH access to the overcloud nodes as **heat-admin**.

COMPREHENSIVE REVIEW

GOAL

Review tasks from *Red Hat OpenStack Administration I: Core Operations for Cloud Operators*

OBJECTIVES

- Review tasks from *Red Hat OpenStack Administration I: Core Operations for Cloud Operators*

SECTIONS

- Comprehensive Review

LAB

- Lab: Deploying an FTP Server
- Lab: Deploying a Web Server
- Lab: Deploying OpenStack and Launching a Stack

COMPREHENSIVE REVIEW

OBJECTIVES

After completing this section, students should have reviewed and refreshed the knowledge and skills learned in *Red Hat OpenStack Administration I: Core Operations for Cloud Operators*.

REVIEWING RED HAT OPENSTACK ADMINISTRATION I: CORE OPERATIONS FOR CLOUD OPERATORS

Before beginning the comprehensive review for this course, students should be comfortable with the topics covered in each chapter.

Do not hesitate to ask the instructor for extra guidance or clarification on these topics.

Chapter 1, *Introducing Launching an Instance*

Launch an instance, and describe the OpenStack architecture and use cases.

- Launch an instance in the dashboard given a preconfigured OpenStack installation.
- Describe the identity environment file and run the unified command-line interface.
- Launch an instance using the command-line interface.
- Describe the OpenStack architecture and use cases.

Chapter 2, *Organizing People and Resources*

Manage projects, users, roles, and quotas.

- Manage OpenStack projects.
- Manage OpenStack user accounts.
- Assign OpenStack user roles and privileges.
- Manage quotas for projects.

Chapter 3, *Describing Cloud Computing*

Describe the changes in technology and processes for cloud computing.

- Describe cloud computing concepts.
- Describe virtual machines and containers.
- Illustrate use cases for Red Hat Enterprise Linux, Red Hat CloudForms, Red Hat Virtualization, and Red Hat OpenStack Platform.
- Verify OpenStack services.

Chapter 4, *Managing Linux Networks*

Manage Linux networks and bridges.

- Describing networking concepts.

- Manage Linux network interfaces.
- Implement Linux bridges.
- Implement Open vSwitch (OVS) bridges.

Chapter 5, Preparing to Deploy an Instance

Manage images, flavors, and private networks in preparation for launching an instance.

- Manage software profiles (images).
- Manage hardware profiles (flavors).
- Manage private networks.

Chapter 6, Deploying an Instance

Launch and verify an instance.

- Launch an instance to illustrate the minimum OpenStack resource preparation required.
- Verify the functionality of an instance.

Chapter 7, Managing Block Storage

Manage ephemeral and persistent block storage.

- Describe cloud storage architecture and features.
- Manage ephemeral block storage.
- Manage persistent block storage.
- Manage snapshots.
- Manage persistent root disks.

Chapter 8, Managing Object Storage

Manage Object Storage

- Describe objects and object storage architecture.
- Manage objects.

Chapter 9, Preparing to Deploy an Instance with Public Access

Manage external networks and security in preparation for launching an instance with public access.

- Manage external networks
- Manage OpenStack routers
- Manage floating IP addresses
- Manage SSH key pairs and security groups

Chapter 10, Deploying an Instance with Public Access

Launch and verify an instance with public access.

- Launch an instance with the additional OpenStack resources required to have public access.
- Verify an instance with public access.

- Manage multi-tenant networking.

Chapter 11, *Customizing Instances*

Customize an instance with cloud-init.

- Customize an instance with cloud-init.
- Verify instance customization.

Chapter 12, *Deploying Scalable Stacks*

Deploy a stack and configure Auto Scaling.

- Analyze OpenStack metrics for use in Auto Scaling.
- Deploy a stack.
- Configure stack Auto Scaling.

Chapter 13, *Deploying an OpenStack Overcloud*

Deploy an OpenStack proof of concept using the director UI and provisioning service templates.

- Deploy OpenStack using the director UI and provisioning service templates.
- Verify OpenStack deployment functionality by querying services and launching an instance.

DEPLOYING AN FTP SERVER

In this review, you will deploy an FTP server in OpenStack.

OUTCOMES

You should be able to:

- Launch an instance in your Red Hat OpenStack Platform environment and customize the instance so it is running an FTP server.
- Access the FTP server from **workstation** to confirm the configuration.
- Store a pseudo-folder and a file in an object container.

Before starting this exercise, save any work you want to keep from earlier exercises. Once you have saved any work you want to keep, reset your classroom environment. Refer to the *To Reset Your Environment* paragraph in the section called “Orientation to the Classroom Environment” at the beginning of this guide.

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-s1 setup
```

Instructions

In this comprehensive review, you will deploy an FTP server in your environment. The architecture will be comprised of an external network and a project network, a new privileged user and a non-privileged user, and a set of new security rules to allow FTP access to the instance. A floating IP address will be associated with the instance to permit external connectivity. A volume will be used by the instance to store the FTP server data. Finally, you will create an object container to store a pseudo-folder and a file.

The environment already provides the following resources for you:

- The **admin** administrative account with **redhat** as the password.
- The dashboard is available at **http://dashboard.overcloud.example.com**.
- An image: **rhel7**
- A flavor: **default**
- A project: **production**

Configure your environment to meet the following requirements:

- Create a user for the **production** project according to the following information.

User Settings

TYPE	VALUE
User name	operator1
User project	production
User roles	_member_ and swiftoperator

- Create a privileged user for the **production** project.

Privileged User Settings

TYPE	VALUE
User name	architect1
User project	production
User role	admin

- Create a new network for the **production** project.

Project Network Settings

NETWORK OBJECT	VALUE
Project network	<ul style="list-style-type: none">• Name: production-network1• Project: production
Project subnet	<ul style="list-style-type: none">• Name: production-subnet1• Network address: 192.168.1.0/24• DHCP Range: 192.168.1.2 to 192.168.1.254• DNS server: 172.25.250.254• DHCP: Enabled• IP version: IPv4

- Create a new cloud provider external network.

External Network Settings

NETWORK OBJECT	VALUE
External network	<ul style="list-style-type: none"> Name: provider-datacentre External network: yes Shared: yes Provider network type: flat Provider physical network: datacentre
External subnet	<ul style="list-style-type: none"> Name: provider-subnet-172.25.250 Network address: 172.25.250.0/24 Range: 172.25.250.101 to 172.25.250.189 Gateway IP: 172.25.250.254 DNS server: 172.25.250.254 DHCP: Disabled IP version: IPv4

- Create a new router for the **production** project.

Router Settings

TYPE	VALUE
Router name	production-router1
External network	provider-datacentre
Internal interface	production-subnet1 subnet

- Define a new security group, **default**, that you will attach to the instance. The security group must contain three rules, for SSH, ICMP, and FTP access.

Security Group Settings

TYPE	VALUE
Security group name	default
Project	production
Security group rules	<ul style="list-style-type: none"> SSH traffic (port 22) from all networks FTP traffic (port range 20:21) from all networks ICMP traffic from all networks

- Define the new key pair **example-keypair1** in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect that file.

- Launch an instance in the **production** project using the following information.

Instance Settings

TYPE	VALUE
Instance name	production-server1
Project	production
Flavor	default
Source image	rhe17
Security group	default
Key pair	example-keypair1
Network	production-network1
Customization script	<pre>#cloud-config runcmd: - echo "Instance customized by cloud-init" > /etc/motd packages: - vsftpd</pre>

- Allocate a floating IP address from the **provider-datacentre** network and associate it with the **production-server1** instance.
- Create and attach a persistent volume to the **production-server1** instance using the following information.

Volume Settings

TYPE	VALUE
Volume name	production-volume1
Project	production
Size	1 GB
Partition settings (from within the instance)	<ul style="list-style-type: none"> • Partition type: primary • Partition number: 1 • Partition size: the whole disk • Partition format type: XFS • Mount point: /var/ftp/pub

- Log in to the **production-server1** instance and configure **vsftpd** as follows:

- As **root**, run `setsebool -P vsftpd_full_access on` to configure SELinux for full FTP access.
 - Add `anon_upload_enable=YES` at the end of `/etc/vsftpd/vsftpd.conf` to allow FTP anonymous uploads.
 - As **root**, run `chown ftp:ftp /var/ftp/pub/` to set the ownership of the FTP upload directory.
 - As **root**, run `systemctl enable vsftpd ; systemctl restart vsftpd` to enable and restart the **vsftpd** service.
- Install an FTP client on **workstation** (the `ftp` package provides such a client). Use this FTP client to test your server by storing a file under **pub**. For this test, log in as the **anonymous** FTP user and disable the passive mode (**passive** command).
 - In preparation for the volume snapshot, unmount and detach the **production-volume1** volume from **production-server1**. Take a snapshot of the volume and name it **production-volume1-snap**.
 - Create a new volume named **production-volume2** from the **production-volume1-snap** snapshot.
 - Associate the two volumes, **production-volume1** and **production-volume2**, to the **production-server1** instance.
 - In **production-server1**, mount the **production-volume1** volume under `/var/ftp/pub` and restart the **vsftpd** service.
 - Create a container named **production-container1** in the **production** project. In that new container, store the `/usr/share/anaconda/pixmaps/sidebar-bg.png` file in the **pictures** pseudo-folder.

Evaluation

As the **student** user on **workstation**, run the `lab compreview-s1` script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-s1 grade
```

Cleanup

From **workstation**, run the `lab compreview-s1 cleanup` command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-s1 cleanup
```

This concludes the lab.

DEPLOYING AN FTP SERVER

In this review, you will deploy an FTP server in OpenStack.

OUTCOMES

You should be able to:

- Launch an instance in your Red Hat OpenStack Platform environment and customize the instance so it is running an FTP server.
- Access the FTP server from **workstation** to confirm the configuration.
- Store a pseudo-folder and a file in an object container.

Before starting this exercise, save any work you want to keep from earlier exercises. Once you have saved any work you want to keep, reset your classroom environment. Refer to the *To Reset Your Environment* paragraph in ??? at the beginning of this guide.

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-s1 setup
```

Instructions

In this comprehensive review, you will deploy an FTP server in your environment. The architecture will be comprised of an external network and a project network, a new privileged user and a non-privileged user, and a set of new security rules to allow FTP access to the instance. A floating IP address will be associated with the instance to permit external connectivity. A volume will be used by the instance to store the FTP server data. Finally, you will create an object container to store a pseudo-folder and a file.

The environment already provides the following resources for you:

- The **admin** administrative account with **redhat** as the password.
- The dashboard is available at **http://dashboard.overcloud.example.com**.
- An image: **rhe17**
- A flavor: **default**
- A project: **production**

Configure your environment to meet the following requirements:

- Create a user for the **production** project according to the following information.

User Settings

TYPE	VALUE
User name	operator1
User project	production
User roles	_member_ and swiftoperator

- Create a privileged user for the **production** project.

Privileged User Settings

TYPE	VALUE
User name	architect1
User project	production
User role	admin

- Create a new network for the **production** project.

Project Network Settings

NETWORK OBJECT	VALUE
Project network	<ul style="list-style-type: none">• Name: production-network1• Project: production
Project subnet	<ul style="list-style-type: none">• Name: production-subnet1• Network address: 192.168.1.0/24• DHCP Range: 192.168.1.2 to 192.168.1.254• DNS server: 172.25.250.254• DHCP: Enabled• IP version: IPv4

- Create a new cloud provider external network.

External Network Settings

NETWORK OBJECT	VALUE
External network	<ul style="list-style-type: none"> Name: provider-datacentre External network: yes Shared: yes Provider network type: flat Provider physical network: datacentre
External subnet	<ul style="list-style-type: none"> Name: provider-subnet-172.25.250 Network address: 172.25.250.0/24 Range: 172.25.250.101 to 172.25.250.189 Gateway IP: 172.25.250.254 DNS server: 172.25.250.254 DHCP: Disabled IP version: IPv4

- Create a new router for the **production** project.

Router Settings

TYPE	VALUE
Router name	production-router1
External network	provider-datacentre
Internal interface	production-subnet1 subnet

- Define a new security group, **default**, that you will attach to the instance. The security group must contain three rules, for SSH, ICMP, and FTP access.

Security Group Settings

TYPE	VALUE
Security group name	default
Project	production
Security group rules	<ul style="list-style-type: none"> SSH traffic (port 22) from all networks FTP traffic (port range 20:21) from all networks ICMP traffic from all networks

- Define the new key pair **example-keypair1** in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect that file.
- Launch an instance in the **production** project using the following information.

Instance Settings

TYPE	VALUE
Instance name	production-server1
Project	production
Flavor	default
Source image	rhel7
Security group	default
Key pair	example-keypair1
Network	production-network1
Customization script	<pre>#cloud-config runcmd: - echo "Instance customized by cloud-init" > /etc/motd packages: - vsftpd</pre>

- Allocate a floating IP address from the **provider-datacentre** network and associate it with the **production-server1** instance.
- Create and attach a persistent volume to the **production-server1** instance using the following information.

Volume Settings

TYPE	VALUE
Volume name	production-volume1
Project	production
Size	1GB
Partition settings (from within the instance)	<ul style="list-style-type: none"> • Partition type: primary • Partition number: 1 • Partition size: the whole disk • Partition format type: XFS • Mount point: /var/ftp/pub

- Log in to the **production-server1** instance and configure **vsftpd** as follows:

- As **root**, run **setsebool -P ftpd_full_access on** to configure SELinux for full FTP access.
- Add **anon_upload_enable=YES** at the end of **/etc/vsftpd/vsftpd.conf** to allow FTP anonymous uploads.
- As **root**, run **chown ftp:ftp /var/ftp/pub/** to set the ownership of the FTP upload directory.
- As **root**, run **systemctl enable vsftpd ; systemctl restart vsftpd** to enable and restart the **vsftpd** service.
- Install an FTP client on **workstation** (the **ftp** package provides such a client). Use this FTP client to test your server by storing a file under **pub**. For this test, log in as the **anonymous** FTP user and disable the passive mode (**passive** command).
- In preparation for the volume snapshot, unmount and detach the **production-volume1** volume from **production-server1**. Take a snapshot of the volume and name it **production-volume1-snap**.
- Create a new volume named **production-volume2** from the **production-volume1-snap** snapshot.
- Associate the two volumes, **production-volume1** and **production-volume2**, to the **production-server1** instance.
- In **production-server1**, mount the **production-volume1** volume under **/var/ftp/pub** and restart the **vsftpd** service.
- Create a container named **production-container1** in the **production** project. In that new container, store the **/usr/share/anaconda/pixmaps/sidebar-bg.png** file in the **pictures** pseudo-folder.

- Create the **operator1** user in the **production** project.

- Source the identity environment file for the **admin** administrative account, create the **operator1** user, and give the user the **_member_** and **swiftoperator** roles in the **production** project.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack user create --project production \
--password redhat operator1
+-----+
| Field          | Value           |
+-----+
| default_project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| domain_id      | default          |
| enabled         | True            |
| id              | 7d063fb0138349948a2611956bafb868 |
| name            | operator1        |
| options          | {}               |
| password_expires_at | None             |
+-----+
[student@workstation ~]$ openstack role add --project production \
--user operator1 _member_
[student@workstation ~]$ openstack role add --project production \
--user operator1 swiftoperator
```

- 1.2. Create the identity environment file for the new user by copying the **/home/student/admin-rc** file to **/home/student/operator1-production-rc** and updating that file.

```
[student@workstation ~]$ cp ~/admin-rc ~/operator1-production-rc
[student@workstation ~]$ vim ~/operator1-production-rc
export CLOUDPROMPT_ENABLED=0
export OS_NO_CACHE=True
export COMPUTE_API_VERSION=1.1

export OS_USERNAME=operator1

export no_proxy=,172.25.250.50,172.25.249.50
export OS_USER_DOMAIN_NAME=Default
export OS_VOLUME_API_VERSION=3
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://172.25.250.50:5000//v3
export NOVA_VERSION=1.1
export OS_IMAGE_API_VERSION=2
export OS_PASSWORD=redhat
export OS_PROJECT_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3

export OS_PROJECT_NAME=production

export OS_AUTH_TYPE=password
export PYTHONWARNINGS="ignore:Certificate has no, ignore:A true SSLContext object
is not available"

export PS1='[\u@\h \w(operator1-production)]\$ '

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
    export PS1=${PS1:-"}
    export PS1=$OS_CLOUDNAME:+("($OS_CLOUDNAME)")\$PS1
    export CLOUDPROMPT_ENABLED=1
fi
```



NOTE

Pay attention to the parameters mentioned in bold in the above screen.

2. Create the **architect1** privileged user in the **production** project.
 - 2.1. Source the identity environment file for the **admin** administrative account, create the **architect1** user, and give the user the **admin** role in the **production** project.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack user create --project production \
--password redhat architect1
+-----+-----+
| Field          | Value           |
+-----+-----+
| default_project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| domain_id      | default         |
```

```

| enabled           | True
| id               | aa3b2365a3b947ef91b37aba64b31d91 |
| name             | architect1
| options          | {}
| password_expires_at | None
+-----+
[student@workstation ~]$ openstack role add --project production \
--user architect1 admin

```

- 2.2. Create the identity environment file for the new user by copying the **/home/student/admin-rc** file to **/home/student/architect1-production-rc** and updating that file.

```

[student@workstation ~]$ cp ~/admin-rc ~/architect1-production-rc
[student@workstation ~]$ vim ~/architect1-production-rc
export CLOUDPROMPT_ENABLED=0
export OS_NO_CACHE=True
export COMPUTE_API_VERSION=1.1

export OS_USERNAME=architect1

export no_proxy=,172.25.250.50,172.25.249.50
export OS_USER_DOMAIN_NAME=Default
export OS_VOLUME_API_VERSION=3
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://172.25.250.50:5000//v3
export NOVA_VERSION=1.1
export OS_IMAGE_API_VERSION=2
export OS_PASSWORD=redhat
export OS_PROJECT_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3

export OS_PROJECT_NAME=production

export OS_AUTH_TYPE=password
export PYTHONWARNINGS="ignore:Certificate has no, ignore:A true SSLContext object
is not available"

export PS1='[\u0@\h \W(architect1-production)]\$ '

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
    export PS1=${PS1:-"}
    export PS1=$OS_CLOUDNAME:+("$OS_CLOUDNAME")$PS1
    export CLOUDPROMPT_ENABLED=1
fi

```

NOTE

Pay attention to the parameters mentioned in bold in the above screen.

3. Create the project **production-network1** network and the **production-subnet1** subnet for the **production** project.

- 3.1. Source the identity environment file for the **operator1** account and create the **production-network1** network.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack network create \
production-network1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T10:37:57Z |
| description | |
| dns_domain | None |
| id | 7fdc48c7-6f12-4f00-8e70-b0db418ec8ee |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1450 |
| name | production-network1 |
| port_security_enabled | True |
| project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| provider:network_type | None |
| provider:physical_network | None |
| provider:segmentation_id | None |
| qos_policy_id | None |
| revision_number | 3 |
| router:external | Internal |
| segments | None |
| shared | False |
| status | ACTIVE |
| subnets | |
| tags | |
| updated_at | 2018-06-21T10:37:57Z |
+-----+
```

- 3.2. Create the **production-subnet1** subnet.

```
[student@workstation ~(operator1-production)]$ openstack subnet create \
--subnet-range 192.168.1.0/24 \
--dhcp \
--allocation-pool start=192.168.1.2,end=192.168.1.254 \
--dns-nameserver 172.25.250.254 \
--network production-network1 \
production-subnet1
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 192.168.1.2-192.168.1.254 |
| cidr | 192.168.1.0/24 |
| created_at | 2018-06-21T10:42:00Z |
+-----+
```

description	
dns_nameservers	172.25.250.254
enable_dhcp	True
gateway_ip	192.168.1.1
host_routes	
id	754c622c-6258-4e88-940f-0ae30c39a1a2
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	production-subnet1
network_id	7fdc48c7-6f12-4f00-8e70-b0db418ec8ee
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-06-21T10:42:00Z

4. Create the external **provider-datacentre** provider network and the **provider-subnet-172.25.250** subnet.

- 4.1. Source the identity environment file for the **admin** administrative account and create the **provider-datacentre** network.

[student@workstation ~]\$ source ~/admin-rc	
[student@workstation ~]\$ openstack network create \	
--external \	
--share \	
--provider-network-type flat \	
--provider-physical-network datacentre \	
provider-datacentre	
+-----+-----+-----+	
Field	Value
+-----+-----+-----+	
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-21T10:47:44Z
description	
dns_domain	None
id	2256afc6-4c24-4942-b419-1d975d06d18a
ipv4_address_scope	None
ipv6_address_scope	None
is_default	False
is_vlan_transparent	None
mtu	1500
name	provider-datacentre
port_security_enabled	True
project_id	50f29df5b0144794a4db5dad1eb12d9a
provider:network_type	flat
provider:physical_network	datacentre
provider:segmentation_id	None
qos_policy_id	None
revision_number	6

router:external	External	
segments	None	
shared	True	
status	ACTIVE	
subnets		
tags		
updated_at	2018-06-21T10:47:45Z	

4.2. Create the **provider-subnet-172.25.250** subnet.

```
[student@workstation ~]$ openstack subnet create \
--no-dhcp \
--subnet-range 172.25.250.0/24 \
--gateway 172.25.250.254 \
--dns-nameserver 172.25.250.254 \
--allocation-pool start=172.25.250.101,end=172.25.250.189 \
--network provider-datacentre \
provider-subnet-172.25.250
+-----+-----+
| Field          | Value           |
+-----+-----+
| allocation_pools | 172.25.250.101-172.25.250.189 |
| cidr          | 172.25.250.0/24 |
| created_at     | 2018-06-21T10:52:51Z |
| description     |                   |
| dns_nameservers | 172.25.250.254 |
| enable_dhcp     | False            |
| gateway_ip      | 172.25.250.254 |
| host_routes     |                   |
| id              | 3cfac2f2-a200-4264-acfb-28f0e96d2c3d |
| ip_version       | 4                |
| ipv6_address_mode | None             |
| ipv6_ra_mode     | None             |
| name            | provider-subnet-172.25.250 |
| network_id       | 2256afc6-4c24-4942-b419-1d975d06d18a |
| project_id       | 50f29df5b0144794a4db5dad1eb12d9a |
| revision_number  | 0                |
| segment_id       | None             |
| service_types    |                   |
| subnetpool_id    | None             |
| tags             |                   |
| updated_at       | 2018-06-21T10:52:51Z |
+-----+-----+
```

5. Create the **production-router1** router in the **production** project. Connect it to the **production-subnet1** private subnet and define it as a gateway for the **provider-datacentre** external network.

5.1. Source the identity environment file for the **operator1** account and create the **production-router1** router.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~]$(operator1-production)]$ openstack router create \
production-router1
+-----+-----+
```

Field	Value
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-21T11:36:01Z
description	
distributed	False
external_gateway_info	None
flavor_id	None
ha	False
id	74cc9507-f70a-4e39-bd6c-1f3289e54b33
name	production-router1
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
revision_number	1
routes	
status	ACTIVE
tags	
updated_at	2018-06-21T11:36:01Z

- 5.2. Connect the **production-router1** router to the **production-subnet1** private subnet. Define it as a gateway for the **provider-datacentre** external network.

```
[student@workstation ~-(operator1-production)]$ openstack router add subnet \
production-router1 production-subnet1
[student@workstation ~-(operator1-production)]$ openstack router set \
--external-gateway provider-datacentre production-router1
```

6. Define the **default** security group in the **production** project. Add rules for SSH, ICMP, and FTP.

- 6.1. Create the **default** security group.

```
[student@workstation ~-(operator1-production)]$ openstack security group \
create default
Error while executing command: HttpException: Unknown error, {"NeutronError": \
{"message": "Default security group already exists.", "type": \
"SecurityGroupDefaultAlreadyExists", "detail": ""}}
```

Notice that the **default** security group already exists because all the projects get a default security group, named **default**, at creation time.

- 6.2. Add the rules to the security group:

- SSH traffic (port 22) from all networks
- FTP traffic (port range 20:21) from all networks
- ICMP traffic from all networks

```
[student@workstation ~-(operator1-production)]$ openstack security group rule \
create --proto tcp --dst-port 22:22 default
+-----+-----+
| Field          | Value           |
+-----+-----+
| created_at     | 2018-06-21T11:49:45Z |
```

```

| description      | |
| direction       | ingress
| ether_type      | IPv4
| id              | 9e955c89-3835-4170-a61e-2e50d0176b20
| name            | None
| port_range_max | 22
| port_range_min | 22
| project_id     | 83aaaf87d7d0b4f49a27b7c7e597d26da
| protocol        | tcp
| remote_group_id| None
| remote_ip_prefix| 0.0.0.0/0
| revision_number| 0
| security_group_id| 694e9d59-1c53-40a8-82bf-11c5c169460d
| updated_at      | 2018-06-21T11:49:45Z
+-----+
[student@workstation ~ (operator1-production)]$ openstack security group rule \
create --proto tcp --dst-port 20:21 default
+-----+
| Field          | Value
+-----+
| created_at    | 2018-06-21T11:50:55Z
| description    |
| direction      | ingress
| ether_type     | IPv4
| id             | 792b9f3a-334c-4c1b-8cad-2ae77e425454
| name           | None
| port_range_max| 21
| port_range_min| 20
| project_id    | 83aaaf87d7d0b4f49a27b7c7e597d26da
| protocol       | tcp
| remote_group_id| None
| remote_ip_prefix| 0.0.0.0/0
| revision_number| 0
| security_group_id| 694e9d59-1c53-40a8-82bf-11c5c169460d
| updated_at     | 2018-06-21T11:50:55Z
+-----+
[student@workstation ~ (operator1-production)]$ openstack security group rule \
create --proto icmp default
+-----+
| Field          | Value
+-----+
| created_at    | 2018-06-21T11:51:57Z
| description    |
| direction      | ingress
| ether_type     | IPv4
| id             | c5aa51d9-70b3-4c72-a7a6-95fe1997d433
| name           | None
| port_range_max| None
| port_range_min| None
| project_id    | 83aaaf87d7d0b4f49a27b7c7e597d26da
| protocol       | icmp
| remote_group_id| None
| remote_ip_prefix| 0.0.0.0/0
| revision_number| 0
| security_group_id| 694e9d59-1c53-40a8-82bf-11c5c169460d
| updated_at     | 2018-06-21T11:51:57Z
+-----+

```

7. Create the **example-keypair1** key pair in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect it.

```
[student@workstation ~-(operator1-production)]$ openstack keypair create \
example-keypair1 > ~/.ssh/example-keypair1
[student@workstation ~-(operator1-production)]$ chmod 600 ~/.ssh/example-keypair1
```

8. Create the **cloud-init** customization script and launch the **production-server1** instance in the **production** project.

- 8.1. Create the customization script in a file.

```
[student@workstation ~-(operator1-production)]$ vim ./script
#cloud-config
runcmd:
- echo "Instance customized by cloud-init" > /etc/motd

packages:
- vsftpd
```



WARNING

Be careful with indentation. You can only use space characters; do not use tab characters. Syntax errors may lead to instance misconfiguration.

- 8.2. Launch the **production-server1** instance.

```
[student@workstation ~-(operator1-production)]$ openstack server create \
--flavor default \
--image rhel7 \
--security-group default \
--key-name example-keypair1 \
--nic net-id=production-network1 \
--user-data ./script \
--wait \
production-server1
+-----+
| Field | Value |
+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2018-06-21T12:08:49.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4 | |
| accessIPv6 | |
| addresses | production-network1=192.168.1.P |
| adminPass | Mq9jyxVfgfiw |
| config_drive | |
| created | 2018-06-21T12:08:03Z |
| flavor | default (f1821919-11ee-4a7f-804e-e01d0a996be8) |
```

hostId	7b05...0fee
id	1a3ca00d-8d9c-44a1-848f-b392dfe0bd1a
image	rhel7 (ca945799-f778-45f0-938f-d72a998ee959)
key_name	example-keypair1
name	production-server1
progress	0
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
properties	
security_groups	name='default'
status	ACTIVE
updated	2018-06-21T12:08:49Z
user_id	7d063fb0138349948a2611956bafb868
volumes_attached	
+-----+-----+	+-----+

9. Allocate a floating IP address from the **provider-datacentre** network and associate it with the **production-server1** instance.

- 9.1. Allocate a floating IP address to the **production** project.

[student@workstation ~ (operator1-production)]\$ openstack floating ip create \ provider-datacentre	
Field	Value
created_at	2018-06-21T12:14:23Z
description	
fixed_ip_address	None
floating_ip_address	172.25.250.N
floating_network_id	2256afc6-4c24-4942-b419-1d975d06d18a
id	4000ad64-8295-4f56-8043-29ffffef66781
name	172.25.250.N
port_id	None
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
qos_policy_id	None
revision_number	0
router_id	None
status	DOWN
subnet_id	None
updated_at	2018-06-21T12:14:23Z
+-----+-----+	+-----+

Take a note of your floating IP address, **172.25.250.N** in the previous output. You will use it in the following steps.

- 9.2. Associate the new floating IP address with the **production-server1** instance.

```
[student@workstation ~ (operator1-production)]$ openstack server add \
floating ip production-server1 172.25.250.N
```

- 9.3. Use the **openstack server show** command to verify the association.

[student@workstation ~ (operator1-production)]\$ openstack server show \ production-server1 -c addresses	
Field	Value
+-----+-----+	+-----+

```
+-----+  
| addresses | production-network1=192.168.1.P, 172.25.250.N |  
+-----+
```

10. Create and attach the **production-volume1** persistent volume to the **production-server1** instance. Log in to **production-server1** and partition, format, and mount the new volume.

10.1. Create the **production-volume1** volume.

```
[student@workstation ~ (operator1-production)]$ openstack volume create \  
--size 1 production-volume1  
+-----+  
| Field          | Value           |  
+-----+  
| attachments    | []              |  
| availability_zone | nova            |  
| bootable        | false            |  
| consistencygroup_id | None            |  
| created_at      | 2018-06-21T12:21:35.000000 |  
| description     | None            |  
| encrypted       | False           |  
| id              | 00532980-6f2a-401d-96a1-8767aee46177 |  
| multiattach     | False           |  
| name            | production-volume1 |  
| properties      |                 |  
| replication_status | None            |  
| size            | 1               |  
| snapshot_id     | None            |  
| source_volid    | None            |  
| status          | creating        |  
| type            | None            |  
| updated_at      | None            |  
| user_id         | 7d063fb0138349948a2611956bafb868 |  
+-----+
```

10.2. When the volume becomes **available**, attach it to the **production-server1** instance.

```
[student@workstation ~ (operator1-production)]$ openstack volume list  
+-----+-----+-----+-----+  
| ID      | Name          | Status   | Size | Attached to |  
+-----+-----+-----+-----+  
| 0053...6177 | production-volume1 | available | 1 |           |  
+-----+-----+-----+-----+  
[student@workstation ~ (operator1-production)]$ openstack server add volume \  
production-server1 production-volume1
```

10.3. Use the **openstack volume list** command again to verify your work.

```
[student@workstation ~ (operator1-production)]$ openstack volume list \  
-c Name -c Status -c "Attached to"  
+-----+-----+-----+  
| Name          | Status | Attached to           |  
+-----+-----+-----+  
| production-volume1 | in-use | Attached to production-server1 on /dev/vdb |
```

- 10.4. Log in with SSH to the **production-server1** instance as **cloud-user** and partition, format, and mount the **production-volume1** volume. When done, log out from **production-server1**.

```
[student@workstation ~(operator1-production)]$ ssh -i ~/.ssh/example-keypair1 \
cloud-user@172.25.250.N
Instance customized by cloud-init
[cloud-user@production-server1 ~]$ sudo -i
[root@production-server1 ~]# fdisk /dev/vdb
Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.

Device does not contain a recognized partition table
Building a new DOS disklabel with disk identifier 0x24fcf997.

Command (m for help): n
Partition type:
   p   primary (0 primary, 0 extended, 4 free)
   e   extended
Select (default p): p
Partition number (1-4, default 1): 1
First sector (2048-2097151, default 2048): Enter
Using default value 2048
Last sector, +sectors or +size{K,M,G} (2048-2097151, default 2097151): Enter
Using default value 2097151
Partition 1 of type Linux and of size 1023 MiB is set

Command (m for help): w
The partition table has been altered!

Calling ioctl() to re-read partition table.
Syncing disks.
[root@production-server1 ~]# mkfs.xfs /dev/vdb1
...output omitted...
[root@production-server1 ~]# mount /dev/vdb1 /var/ftp/pub
[root@production-server1 ~]# exit
[cloud-user@production-server1 ~]$ exit
[student@workstation ~(operator1-production)]$
```

11. Configure the **vsftpd** FTP server on **production-server1**. When done, log out from **production-server1**.

```
[student@workstation ~(operator1-production)]$ ssh -i ~/.ssh/example-keypair1 \
cloud-user@172.25.250.N
Instance customized by cloud-init
[cloud-user@production-server1 ~]$ sudo -i
[root@production-server1 ~]# setsebool -P ftpd_full_access on
[root@production-server1 ~]# echo anon_upload_enable=YES >> \
/etc/vsftpd/vsftpd.conf
[root@production-server1 ~]# chown -R ftp:ftp /var/ftp/pub
[root@production-server1 ~]# systemctl enable vsftpd
```

```
Created symlink from /etc/systemd/system/multi-user.target.wants/vsftpd.service
to /usr/lib/systemd/system/vsftpd.service.
[root@production-server1 ~]# systemctl restart vsftpd
[root@production-server1 ~]# exit
[cloud-user@production-server1 ~]$ exit
[student@workstation ~(operator1-production)]$
```

12. Verify the FTP server functionality from **workstation**.

- 12.1. On **workstation**, create a text file, **test_file.txt** for example, and add some content to it.

```
[student@workstation ~(operator1-production)]$ echo "This is my file" > \
test_file.txt
```

- 12.2. Install the **ftp** package to get an FTP client.

```
[student@workstation ~(operator1-production)]$ sudo yum -y install ftp
[sudo] password for student: student
...output omitted...
```

- 12.3. Open an FTP session to the FTP server, disable passive mode, and deposit your **test_file.txt** file under **pub**. When done, close the FTP session.

```
[student@workstation ~(operator1-production)]$ ftp 172.25.250.N
Connected to 172.25.250.N (172.25.250.N).
220 (vsFTPd 3.0.2)
Name (172.25.250.N:student): anonymous
331 Please specify the password.
Password: Enter
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> passive
Passive mode off.
ftp> cd pub
250 Directory successfully changed.
ftp> put test_file.txt
local: test_file.txt remote: test_file.txt
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
16 bytes sent in 5.6e-05 secs (285.71 Kbytes/sec)
ftp> quit
221 Goodbye.
[student@workstation ~(operator1-production)]$
```

13. Unmount and detach the **production-volume1** volume from **production-server1**, and take a snapshot of the volume. The snapshot name is **production-volume1-snap**.

- 13.1. Log in to the **production-server1** instance and unmount the volume. When done, log out from **production-server1**.

```
[student@workstation ~(operator1-production)]$ ssh -i ~/.ssh/example-keypair1 \
cloud-user@172.25.250.N
Instance customized by cloud-init
```

```
[cloud-user@production-server1 ~]$ sudo umount /var/ftp/pub  
[cloud-user@production-server1 ~]$ exit  
[student@workstation ~](operator1-production)]$
```

13.2. Detach the **production-volume1** volume and take a snapshot of the volume.

```
[student@workstation ~](operator1-production)]$ openstack server remove volume \  
production-server1 production-volume1  
[student@workstation ~](operator1-production)]$ openstack volume snapshot \  
create --volume production-volume1 production-volume1-snap  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| created_at | 2018-06-21T13:25:03.386367 |  
| description | None |  
| id | 84d977f9-ddd8-450a-9c93-99990a357c21 |  
| name | production-volume1-snap |  
| properties | |  
| size | 1 |  
| status | creating |  
| updated_at | None |  
| volume_id | 00532980-6f2a-401d-96a1-8767aeee46177 |  
+-----+-----+
```

13.3. Wait for the snapshot to complete.

```
[student@workstation ~](operator1-production)]$ openstack volume snapshot list  
+-----+-----+-----+-----+  
| ID | Name | Description | Status | Size |  
+-----+-----+-----+-----+  
| 84d9...7c21 | production-volume1-snap | None | available | 1 |  
+-----+-----+-----+-----+
```

14. Create the **production-volume2** volume from the **production-volume1-snap** snapshot. Wait for the volume creation to complete.

```
[student@workstation ~](operator1-production)]$ openstack volume create \  
--snapshot production-volume1-snap production-volume2  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| attachments | [] |  
| availability_zone | nova |  
| bootable | false |  
| consistencygroup_id | None |  
| created_at | 2018-06-21T13:29:08.000000 |  
| description | None |  
| encrypted | False |  
| id | f64b9035-65d7-488f-b702-b2c7bde144fd |  
| multiattach | False |  
| name | production-volume2 |  
| properties | |  
| replication_status | None |  
| size | 1 |  
| snapshot_id | 84d977f9-ddd8-450a-9c93-99990a357c21 |
```

```

| source_volid      | None
| status           | creating
| type             | None
| updated_at       | None
| user_id          | 7d063fb0138349948a2611956bafb868
+-----+
[student@workstation ~-(operator1-production)]$ openstack volume list
+-----+-----+-----+-----+
| ID      | Name           | Status   | Size | Attached to |
+-----+-----+-----+-----+
| f64b...44fd | production-volume2 | available |    1 |           |
| 0053...6177 | production-volume1 | available |    1 |           |
+-----+-----+-----+-----+

```

15. Associate the two volumes, **production-volume1** and **production-volume2**, to the **production-server1** instance.

- 15.1. Add the volumes to the instance.

```

[student@workstation ~-(operator1-production)]$ openstack server add volume \
production-server1 production-volume1
[student@workstation ~-(operator1-production)]$ openstack server add volume \
production-server1 production-volume2

```

- 15.2. Use the **openstack volume list** command to verify your work.

```

[student@workstation ~-(operator1-production)]$ openstack volume list \
-c Name -c Status -c "Attached to"
+-----+-----+-----+
| Name           | Status | Attached to           |
+-----+-----+-----+
| production-volume2 | in-use | Attached to production-server1 on /dev/vdc |
| production-volume1 | in-use | Attached to production-server1 on /dev/vdb |
+-----+-----+-----+

```

16. Log in to **production-server1** and mount the **production-volume1** volume under **/var/ftp/pub**. Restart the **vsftpd** service and log out from **production-server1** when done.

```

[student@workstation ~-(operator1-production)]$ ssh -i ~/.ssh/example-keypair1 \
cloud-user@172.25.250.N
Instance customized by cloud-init
[cloud-user@production-server1 ~]$ sudo -i
[root@production-server1 ~]# mount /dev/vdb1 /var/ftp/pub
[root@production-server1 ~]# systemctl restart vsftpd
[root@production-server1 ~]# exit
[cloud-user@production-server1 ~]$ exit
[student@workstation ~-(operator1-production)]$ 

```

17. Create the **production-container1** container in the **production** project. In that new container, store the **/usr/share/anaconda/pixmaps/sidebar-bg.png** file in the **pictures** pseudo-folder.

- 17.1. Create the **production-container1** container.

```
[student@workstation ~-(operator1-production)]$ openstack container create \

```

```
production-container1
+-----+-----+-----+
| account | container | x-trans-id |
+-----+-----+-----+
| AUTH_83aa...26da | production-container1 | txb3...aaa0 |
+-----+-----+-----+
```

17.2. Prepare the file and folder to store as objects in the container.

```
[student@workstation ~](operator1-production)]$ mkdir pictures
[student@workstation ~](operator1-production)]$ cp \
/usr/share/anaconda/pixmaps/sidebar-bg.png pictures
```

17.3. Store the folder and its file in the **production-container1** container.

```
[student@workstation ~](operator1-production)]$ openstack object create \
production-container1 pictures/sidebar-bg.png
+-----+-----+-----+
| object | container | etag |
+-----+-----+-----+
| pictures/sidebar-bg.png | production-container1 | d88c...f32e |
+-----+-----+-----+
```

Evaluation

As the **student** user on **workstation**, run the **lab compreview-s1** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-s1 grade
```

Cleanup

From **workstation**, run the **lab compreview-s1 cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-s1 cleanup
```

This concludes the lab.

DEPLOYING A WEB SERVER

In this review, you will deploy a web server in OpenStack.

OUTCOMES

You should be able to:

- Launch an instance in your Red Hat OpenStack Platform environment and customize the instance so it is running a web server.
- Access the web server from **workstation** to confirm the configuration.

Before starting this exercise, save any work you want to keep from earlier exercises. Once you have saved any work you want to keep, reset your classroom environment. Refer to the *To Reset Your Environment* paragraph in the section called “Orientation to the Classroom Environment” at the beginning of this guide.

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-s2 setup
```

Instructions

In this comprehensive review, you will deploy a web server in your environment. The architecture will be comprised of an external network and a project network, a new privileged user and a non-privileged user, and a set of new security rules to allow HTTP access to the instance. A floating IP address will be associated with the instance to permit external connectivity.

The environment already provides the following resources for you:

- The **admin** administrative account with **redhat** as the password.
- The dashboard is available at **http://dashboard.overcloud.example.com**.
- An image: **rhe17**
- A flavor: **default**
- A project: **production**

Configure your environment to meet the following requirements:

- Create a user for the **production** project according to the following information.

User Settings

TYPE	VALUE
User name	operator1

TYPE	VALUE
User project	production
User role	_member_

- Create a privileged user for the **production** project.

Privileged User Settings

TYPE	VALUE
User name	architect1
User project	production
User role	admin

- Create a new network for the **production** project.

Project Network Settings

NETWORK OBJECT	VALUE
Project network	<ul style="list-style-type: none"> • Name: production-network1 • Project: production
Project subnet	<ul style="list-style-type: none"> • Name: production-subnet1 • Network address: 192.168.1.0/24 • DHCP Range: 192.168.1.2 to 192.168.1.254 • DNS server: 172.25.250.254 • DHCP: Enabled • IP version: IPv4

- Create a new cloud provider external network.

External Network Settings

NETWORK OBJECT	VALUE
External network	<ul style="list-style-type: none"> • Name: provider-datacentre • External network: yes • Shared: yes • Provider network type: flat • Provider physical network: datacentre

NETWORK OBJECT	VALUE
External subnet	<ul style="list-style-type: none"> Name: provider-subnet-172.25.250 Network address: 172.25.250.0/24 Range: 172.25.250.101 to 172.25.250.189 Gateway IP: 172.25.250.254 DNS server: 172.25.250.254 DHCP: Disabled IP version: IPv4

- Create a new router for the **production** project.

Router Settings

TYPE	VALUE
Router name	production-router1
External network	provider-datacentre
Internal interface	production-subnet1 subnet

- Define a new security group, **default**, that you will attach to the instance. The security group must contain three rules, for SSH, ICMP, and HTTP access.

Security Group Settings

TYPE	VALUE
Security group name	default
Project	production
Security group rules	<ul style="list-style-type: none"> SSH traffic (port 22) from all networks HTTP traffic (port 80) from all networks ICMP traffic from all networks

- Define the new key pair **example-keypair1** in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect that file.
- Launch an instance in the **production** project using the following information.

Instance Settings

TYPE	VALUE
Instance name	production-server1

TYPE	VALUE
Project	production
Flavor	default
Source image	rhel7
Security group	default
Key pair	example-keypair1
Network	production-network1
Customization script	<pre>#!/bin/bash yum -y install httpd systemctl enable httpd --now cd /var/www/html echo "Hello World!" > index.html</pre>

- Allocate a floating IP address from the **provider-datacentre** network and associate it with the **production-server1** instance.

Evaluation

As the **student** user on **workstation**, run the **lab compreview-s2** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-s2 grade
```

Cleanup

From **workstation**, run the **lab compreview-s2 cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-s2 cleanup
```

This concludes the lab.

DEPLOYING A WEB SERVER

In this review, you will deploy a web server in OpenStack.

OUTCOMES

You should be able to:

- Launch an instance in your Red Hat OpenStack Platform environment and customize the instance so it is running a web server.
- Access the web server from **workstation** to confirm the configuration.

Before starting this exercise, save any work you want to keep from earlier exercises. Once you have saved any work you want to keep, reset your classroom environment. Refer to the *To Reset Your Environment* paragraph in ??? at the beginning of this guide.

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-s2 setup
```

Instructions

In this comprehensive review, you will deploy a web server in your environment. The architecture will be comprised of an external network and a project network, a new privileged user and a non-privileged user, and a set of new security rules to allow HTTP access to the instance. A floating IP address will be associated with the instance to permit external connectivity.

The environment already provides the following resources for you:

- The **admin** administrative account with **redhat** as the password.
- The dashboard is available at **http://dashboard.overcloud.example.com**.
- An image: **rhe17**
- A flavor: **default**
- A project: **production**

Configure your environment to meet the following requirements:

- Create a user for the **production** project according to the following information.

User Settings

TYPE	VALUE
User name	operator1

TYPE	VALUE
User project	production
User role	_member_

- Create a privileged user for the **production** project.

Privileged User Settings

TYPE	VALUE
User name	architect1
User project	production
User role	admin

- Create a new network for the **production** project.

Project Network Settings

NETWORK OBJECT	VALUE
Project network	<ul style="list-style-type: none"> • Name: production-network1 • Project: production
Project subnet	<ul style="list-style-type: none"> • Name: production-subnet1 • Network address: 192.168.1.0/24 • DHCP Range: 192.168.1.2 to 192.168.1.254 • DNS server: 172.25.250.254 • DHCP: Enabled • IP version: IPv4

- Create a new cloud provider external network.

External Network Settings

NETWORK OBJECT	VALUE
External network	<ul style="list-style-type: none"> • Name: provider-datacentre • External network: yes • Shared: yes • Provider network type: flat • Provider physical network: datacentre

NETWORK OBJECT	VALUE
External subnet	<ul style="list-style-type: none"> Name: provider-subnet-172.25.250 Network address: 172.25.250.0/24 Range: 172.25.250.101 to 172.25.250.189 Gateway IP: 172.25.250.254 DNS server: 172.25.250.254 DHCP: Disabled IP version: IPv4

- Create a new router for the **production** project.

Router Settings

TYPE	VALUE
Router name	production-router1
External network	provider-datacentre
Internal interface	production-subnet1 subnet

- Define a new security group, **default**, that you will attach to the instance. The security group must contain three rules, for SSH, ICMP, and HTTP access.

Security Group Settings

TYPE	VALUE
Security group name	default
Project	production
Security group rules	<ul style="list-style-type: none"> SSH traffic (port 22) from all networks HTTP traffic (port 80) from all networks ICMP traffic from all networks

- Define the new key pair **example-keypair1** in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect that file.
- Launch an instance in the **production** project using the following information.

Instance Settings

TYPE	VALUE
Instance name	production-server1

TYPE	VALUE
Project	production
Flavor	default
Source image	rhel7
Security group	default
Key pair	example-keypair1
Network	production-network1
Customization script	<pre>#!/bin/bash yum -y install httpd systemctl enable httpd --now cd /var/www/html echo "Hello World!" > index.html</pre>

- Allocate a floating IP address from the **provider-datacentre** network and associate it with the **production-server1** instance.

1. Create the **operator1** user in the **production** project.

- 1.1. Source the identity environment file for the **admin** administrative account, create the **operator1** user, and give the user the **_member_** role in the **production** project.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack user create --project production \
--password redhat operator1
+-----+-----+
| Field      | Value   |
+-----+-----+
| default_project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| domain_id    | default |
| enabled       | True    |
| id           | 7d063fb0138349948a2611956bafb868 |
| name          | operator1 |
| options        | {}      |
| password_expires_at | None    |
+-----+-----+
[student@workstation ~]$ openstack role add --project production \
--user operator1 _member_
```

- 1.2. Create the identity environment file for the new user by copying the **/home/student/admin-rc** file to **/home/student/operator1-production-rc** and updating that file.

```
[student@workstation ~]$ cp ~/admin-rc ~/operator1-production-rc
[student@workstation ~]$ vim ~/operator1-production-rc
export CLOUDPROMPT_ENABLED=0
export OS_NO_CACHE=True
export COMPUTE_API_VERSION=1.1
```

```

export OS_USERNAME=operator1

export no_proxy=,172.25.250.50,172.25.249.50
export OS_USER_DOMAIN_NAME=Default
export OS_VOLUME_API_VERSION=3
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://172.25.250.50:5000//v3
export NOVA_VERSION=1.1
export OS_IMAGE_API_VERSION=2
export OS_PASSWORD=redhat
export OS_PROJECT_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3

export OS_PROJECT_NAME=production

export OS_AUTH_TYPE=password
export PYTHONWARNINGS="ignore:Certificate has no, ignore:A true SSLContext object
is not available"

export PS1='[\u@\$h \w(operator1-production)]\$ '

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
    export PS1=${PS1:-"}
    export PS1=\${OS_CLOUDNAME:+"\$OS_CLOUDNAME"}\$PS1
    export CLOUDPROMPT_ENABLED=1
fi

```



NOTE

Pay attention to the parameters mentioned in bold in the above screen.

2. Create the **architect1** privileged user in the **production** project.
 - 2.1. Source the identity environment file for the **admin** administrative account, create the **architect1** user, and give the user the **admin** role in the **production** project.

```

[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack user create --project production \
--password redhat architect1
+-----+-----+
| Field          | Value           |
+-----+-----+
| default_project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| domain_id      | default         |
| enabled         | True            |
| id              | aa3b2365a3b947ef91b37aba64b31d91 |
| name            | architect1     |
| options          | {}              |
| password_expires_at | None           |
+-----+-----+
[student@workstation ~]$ openstack role add --project production \

```

```
--user architect1 admin
```

- 2.2. Create the identity environment file for the new user by copying the **/home/student/admin-rc** file to **/home/student/architect1-production-rc** and updating that file.

```
[student@workstation ~]$ cp ~/admin-rc ~/architect1-production-rc
[student@workstation ~]$ vim ~/architect1-production-rc
export CLOUDPROMPT_ENABLED=0
export OS_NO_CACHE=True
export COMPUTE_API_VERSION=1.1

export OS_USERNAME=architect1

export no_proxy=,172.25.250.50,172.25.249.50
export OS_USER_DOMAIN_NAME=Default
export OS_VOLUME_API_VERSION=3
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://172.25.250.50:5000//v3
export NOVA_VERSION=1.1
export OS_IMAGE_API_VERSION=2
export OS_PASSWORD=redhat
export OS_PROJECT_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3

export OS_PROJECT_NAME=production

export OS_AUTH_TYPE=password
export PYTHONWARNINGS="ignore:Certificate has no, ignore:A true SSLContext object
is not available"

export PS1='[\u@\h \W(architect1-production)]\$ '

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
    export PS1=${PS1:-""}
    export PS1=\${OS_CLOUDNAME:+"\${OS_CLOUDNAME}")}\ $PS1
    export CLOUDPROMPT_ENABLED=1
fi
```



NOTE

Pay attention to the parameters mentioned in bold in the above screen.

3. Create the project **production-network1** network and the **production-subnet1** subnet for the **production** project.
 - 3.1. Source the identity environment file for the **operator1** account and create the **production-network1** network.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack network create \
production-network1
+-----+-----+
| Field | Value |
+-----+-----+
```

admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-21T10:37:57Z
description	
dns_domain	None
id	7fdc48c7-6f12-4f00-8e70-b0db418ec8ee
ipv4_address_scope	None
ipv6_address_scope	None
is_default	False
is_vlan_transparent	None
mtu	1450
name	production-network1
port_security_enabled	True
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
provider:network_type	None
provider:physical_network	None
provider:segmentation_id	None
qos_policy_id	None
revision_number	3
router:external	Internal
segments	None
shared	False
status	ACTIVE
subnets	
tags	
updated_at	2018-06-21T10:37:57Z

3.2. Create the **production-subnet1** subnet.

[student@workstation ~ (operator1-production)]\$ openstack subnet create \
--subnet-range 192.168.1.0/24 \
--dns-nameserver 172.25.250.254 \
--dhcp \
--allocation-pool start=192.168.1.2,end=192.168.1.254 \
--network production-network1 \
production-subnet1
+-----+-----+-----+
Field Value
+-----+-----+-----+
allocation_pools 192.168.1.2-192.168.1.254
cidr 192.168.1.0/24
created_at 2018-06-21T10:42:00Z
description
dns_nameservers 172.25.250.254
enable_dhcp True
gateway_ip 192.168.1.1
host_routes
id 754c622c-6258-4e88-940f-0ae30c39a1a2
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name production-subnet1
network_id 7fdc48c7-6f12-4f00-8e70-b0db418ec8ee
project_id 83aaaf87d7d0b4f49a27b7c7e597d26da

revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-06-21T10:42:00Z

4. Create the external **provider-datacentre** provider network and the **provider-subnet-172.25.250** subnet.

- 4.1. Source the identity environment file for the **admin** administrative account and create the **provider-datacentre** network.

```
[student@workstation ~](operator1-production)]$ source ~/admin-rc
[student@workstation ~]$ openstack network create \
--external \
--share \
--provider-network-type flat \
--provider-physical-network datacentre \
provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T10:47:44Z |
| description | |
| dns_domain | None |
| id | 2256afc6-4c24-4942-b419-1d975d06d18a |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1500 |
| name | provider-datacentre |
| port_security_enabled | True |
| project_id | 50f29df5b0144794a4db5dad1eb12d9a |
| provider:network_type | flat |
| provider:physical_network | datacentre |
| provider:segmentation_id | None |
| qos_policy_id | None |
| revision_number | 6 |
| router:external | External |
| segments | None |
| shared | True |
| status | ACTIVE |
| subnets | |
| tags | |
| updated_at | 2018-06-21T10:47:45Z |
+-----+-----+
```

- 4.2. Create the **provider-subnet-172.25.250** subnet.

```
[student@workstation ~]$ openstack subnet create \
```

```
--no-dhcp \
--subnet-range 172.25.250.0/24 \
--gateway 172.25.250.254 \
--dns-nameserver 172.25.250.254 \
--allocation-pool start=172.25.250.101,end=172.25.250.189 \
--network provider-datacentre \
provider-subnet-172.25.250
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 172.25.250.101-172.25.250.189 |
| cidr | 172.25.250.0/24 |
| created_at | 2018-06-21T10:52:51Z |
| description | |
| dns_nameservers | 172.25.250.254 |
| enable_dhcp | False |
| gateway_ip | 172.25.250.254 |
| host_routes | |
| id | 3cfac2f2-a200-4264-acfb-28f0e96d2c3d |
| ip_version | 4 |
| ipv6_address_mode | None |
| ipv6_ra_mode | None |
| name | provider-subnet-172.25.250 |
| network_id | 2256afc6-4c24-4942-b419-1d975d06d18a |
| project_id | 50f29df5b0144794a4db5dad1eb12d9a |
| revision_number | 0 |
| segment_id | None |
| service_types | |
| subnetpool_id | None |
| tags | |
| updated_at | 2018-06-21T10:52:51Z |
+-----+-----+
```

5. Create the **production-router1** router in the **production** project. Connect it to the **production-subnet1** private subnet and define it as a gateway for the **provider-datacentre** external network.

- 5.1. Source the identity environment file for the **operator1** account and create the **production-router1** router.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack router create \
production-router1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T11:36:01Z |
| description | |
| distributed | False |
| external_gateway_info | None |
| flavor_id | None |
| ha | False |
| id | 74cc9507-f70a-4e39-bd6c-1f3289e54b33 |
```

name	production-router1	
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da	
revision_number	1	
routes		
status	ACTIVE	
tags		
updated_at	2018-06-21T11:36:01Z	

- 5.2. Connect the **production-router1** router to the **production-subnet1** private subnet. Define it as a gateway for the **provider-datacentre** external network.

```
[student@workstation ~ (operator1-production)]$ openstack router add subnet \
production-router1 production-subnet1
[student@workstation ~ (operator1-production)]$ openstack router set \
--external-gateway provider-datacentre production-router1
```

6. Define the **default** security group in the **production** project. Add rules for SSH, ICMP, and HTTP.

- 6.1. Create the **default** security group.

```
[student@workstation ~ (operator1-production)]$ openstack security group \
create default
Error while executing command: HttpException: Unknown error, {"NeutronError": \
{"message": "Default security group already exists.", "type": \
"SecurityGroupDefaultAlreadyExists", "detail": ""}}
```

Notice that the **default** security group already exists because all the projects get a default security group, named **default**, at creation time.

- 6.2. Add the rules to the security group:

- SSH traffic (port 22) from all networks
- HTTP traffic (port 80) from all networks
- ICMP traffic from all networks

[student@workstation ~ (operator1-production)]\$ openstack security group rule \ create --proto tcp --dst-port 22:22 default		
Field	Value	
created_at	2018-06-21T11:49:45Z	
description		
direction	ingress	
ether_type	IPv4	
id	9e955c89-3835-4170-a61e-2e50d0176b20	
name	None	
port_range_max	22	
port_range_min	22	
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da	
protocol	tcp	
remote_group_id	None	
remote_ip_prefix	0.0.0.0/0	
revision_number	0	

```

| security_group_id | 694e9d59-1c53-40a8-82bf-11c5c169460d |
| updated_at        | 2018-06-21T11:49:45Z |
+-----+-----+
[student@workstation ~-(operator1-production)]$ openstack security group rule \
create --proto tcp --dst-port 80:80 default
+-----+-----+
| Field      | Value   |
+-----+-----+
| created_at | 2018-06-21T11:50:55Z |
| description |          |
| direction   | ingress |
| ether_type  | IPv4    |
| id          | 792b9f3a-334c-4c1b-8cad-2ae77e425454 |
| name        | None    |
| port_range_max | 80 |
| port_range_min | 80 |
| project_id  | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| protocol    | tcp     |
| remote_group_id | None |
| remote_ip_prefix | 0.0.0.0/0 |
| revision_number | 0 |
| security_group_id | 694e9d59-1c53-40a8-82bf-11c5c169460d |
| updated_at   | 2018-06-21T11:50:55Z |
+-----+-----+
[student@workstation ~-(operator1-production)]$ openstack security group rule \
create --proto icmp default
+-----+-----+
| Field      | Value   |
+-----+-----+
| created_at | 2018-06-21T11:51:57Z |
| description |          |
| direction   | ingress |
| ether_type  | IPv4    |
| id          | c5aa51d9-70b3-4c72-a7a6-95fe1997d433 |
| name        | None    |
| port_range_max | None |
| port_range_min | None |
| project_id  | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| protocol    | icmp    |
| remote_group_id | None |
| remote_ip_prefix | 0.0.0.0/0 |
| revision_number | 0 |
| security_group_id | 694e9d59-1c53-40a8-82bf-11c5c169460d |
| updated_at   | 2018-06-21T11:51:57Z |
+-----+-----+

```

7. Create the **example-keypair1** key pair in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect it.

```

[student@workstation ~-(operator1-production)]$ openstack keypair create \
example-keypair1 > ~/.ssh/example-keypair1
[student@workstation ~-(operator1-production)]$ chmod 600 ~/.ssh/example-keypair1

```

8. Create the customization script and launch the **production-server1** instance in the **production** project.

8.1. Create the customization script in a file.

```
[student@workstation ~(operator1-production)]$ vim ./myscript
#!/bin/bash
yum -y install httpd
systemctl enable httpd --now
cd /var/www/html
echo "Hello World!" > index.html
```

8.2. Launch the **production-server1** instance.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--image rhel7 \
--security-group default \
--key-name example-keypair1 \
--nic net-id=production-network1 \
--user-data ./myscript \
--wait \
production-server1
+-----+
| Field | Value |
+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-AZ:availability_zone | nova |
| OS-EXT-STS:power_state | Running |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| OS-SRV-USG:launched_at | 2018-06-21T12:08:49.000000 |
| OS-SRV-USG:terminated_at | None |
| accessIPv4 |
| accessIPv6 |
| addresses | production-network1=192.168.1.1 |
| adminPass | Mq9jyxVfgfiw |
| config_drive |
| created | 2018-06-21T12:08:03Z |
| flavor | default (f1821919-11ee-4a7f-804e-e01d0a996be8) |
| hostId | 7b05...0fee |
| id | 1a3ca00d-8d9c-44a1-848f-b392dfe0bd1a |
| image | rhel7 (ca945799-f778-45f0-938f-d72a998ee959) |
| key_name | example-keypair1 |
| name | production-server1 |
| progress | 0 |
| project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| properties |
| security_groups | name='default' |
| status | ACTIVE |
| updated | 2018-06-21T12:08:49Z |
| user_id | 7d063fb0138349948a2611956bafb868 |
| volumes_attached |
```

9. Allocate a floating IP address from the **provider-datacentre** network and associate it with the **production-server1** instance.

- 9.1. Allocate a floating IP address to the **production** project.

```
[student@workstation ~-(operator1-production)]$ openstack floating ip create \
provider-datacentre
+-----+-----+
| Field | Value |
+-----+-----+
| created_at | 2018-06-21T12:14:23Z |
| description | |
| fixed_ip_address | None |
| floating_ip_address | 172.25.250.N |
| floating_network_id | 2256afc6-4c24-4942-b419-1d975d06d18a |
| id | 4000ad64-8295-4f56-8043-29ffffef66781 |
| name | 172.25.250.N |
| port_id | None |
| project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| qos_policy_id | None |
| revision_number | 0 |
| router_id | None |
| status | DOWN |
| subnet_id | None |
| updated_at | 2018-06-21T12:14:23Z |
+-----+-----+
```

Take a note of your floating IP address, **172.25.250.N** in the previous output. You will use it in the following steps.

- 9.2. Associate the new floating IP address with the **production-server1** instance.

```
[student@workstation ~-(operator1-production)]$ openstack server add \
floating_ip production-server1 172.25.250.N
```

- 9.3. Use the **openstack server show** command to verify the association.

```
[student@workstation ~-(operator1-production)]$ openstack server show \
production-server1 -c addresses
+-----+-----+
| Field | Value |
+-----+-----+
| addresses | production-network1=192.168.1.P, 172.25.250.N |
+-----+-----+
```

10. Use the **curl** command to test your web server from **workstation**.

```
[student@workstation ~-(operator1-production)]$ curl http://172.25.250.N
Hello World!
```

Evaluation

As the **student** user on **workstation**, run the **lab_compreview-s2** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab compreview-s2 grade
```

Cleanup

From **workstation**, run the **lab compreview-s2 cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab compreview-s2 cleanup
```

This concludes the lab.

DEPLOYING OPENSTACK AND LAUNCHING A STACK

In this review, you will deploy OpenStack using director UI and launch an orchestration stack.

OUTCOMES

You should be able to:

- Create an overcloud plan directory with the director's base templates and include the plan environment file based on a particular deployment architecture.
- Import an overcloud plan using the web UI.
- Deploy the Red Hat OpenStack Platform overcloud using the director UI.
- Create resources for an orchestration stack.
- Launch an orchestration stack.

Confirm the **workstation** and the overcloud's virtual machines are started.

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-s3 setup
```

This command:

- Removes the **overcloud** deployment.
- Ensures the **controller0** and **compute0** bare metal nodes are configured.
- Deletes any existing overcloud plans.

Instructions

In this comprehensive review, you will import an overcloud plan into the Red Hat OpenStack Platform director UI and initiate the deployment using one controller and one compute node. Once the overcloud is deployed, you will use orchestration templates to deploy a stack that auto-scale a web server using the CPU utilization metrics.

The environment already provides the following resources for you:

- The Red Hat OpenStack Platform director UI is available at **https://director-ui.lab.example.com**. Use **admin** as user name and **redhat** as the password to access this web UI.
- The heat templates for director are available in **/usr/share/openstack-tripleo-heat-templates**.

- The **<http://materials.example.com/classroom-plan.tar.gz>** tar file contains a custom **plan-environment.yaml** file, and all the customization required for the Red Hat OpenStack Platform deployment in the classroom environment.
- The **<http://materials.example.com/osp-web.qcow2>** file contains an image of an installed RHEL system with a configured web server.

Configure your environment to meet the following requirements:

- Copy the **/usr/share/openstack-tripleo-heat-templates** directory on **director** in **/home/student/overcloud-poc** on **workstation**. In that directory, apply the customization provided by the **<http://materials.example.com/classroom-plan.tar.gz>** tar file.
- Update the custom **plan-environment.yaml** file in **/home/student/overcloud-poc** to force the overcloud **admin** user password to **redhat**. For that, add **AdminPassword: redhat** to the **parameter_defaults** section.
- Using the director web UI, import the overcloud plan provided by the **/home/student/overcloud-poc** directory. The director web UI is available at **<https://director-ui.lab.example.com>**. Use **admin** as user name and **redhat** as the password to access it.
- Ensure that one **Controller** role and one **Compute** role are assigned to the bare metal nodes.
- Deploy the overcloud. After the overcloud deployment process completes, the Red Hat OpenStack Platform dashboard for the overcloud should be available at **<http://dashboard.overcloud.example.com>**. Create the **admin** identity environment file in **/home/student/admin-rc** on workstation.
- Create the resources that you need later to deploy the autoscaling stack. First, create a flavor in the overcloud according to the following information.

Flavor Settings

TYPE	VALUE
Flavor name	default-no-swap
RAM	1024 MB
Root disk	10 GB
Number of VCPUs	2
Ephemeral disk	0
Swap	0

- Create a public image in the overcloud according to the following information.

Image Settings

TYPE	VALUE
Image name	rhel7-web
Disk image URL	http://materials.example.com/osp-web.qcow2
Protected	yes

TYPE	VALUE
Public	yes
Minimum disk size	10 GB
Minimum ram	1024 MB

- Create the **production** project in the overcloud.
- Create the **heat_stack_owner** role in the overcloud.
- Create a user for the **production** project according to the following information.

User Settings

TYPE	VALUE
User name	operator1
User project	production
User role	_member_, heat_stack_owner

- Create a privileged user for the **production** project.

Privileged User Settings

TYPE	VALUE
User name	architect1
User project	production
User role	admin

- Create a new cloud provider public network.

External Network Settings

NETWORK OBJECT	VALUE
Public network	<ul style="list-style-type: none"> • Name: provider-datacentre • External network: yes • Shared: yes • Provider network type: flat • Provider physical network: datacentre

NETWORK OBJECT	VALUE
Public subnet	<ul style="list-style-type: none"> Name: provider-subnet-172.25.250 Network address: 172.25.250.0/24 Range: 172.25.250.101 to 172.25.250.189 Gateway IP: 172.25.250.254 DNS server: 172.25.250.254 DHCP: Disabled IP version: IPv4

- Create a new network for the **production** project.

Project Network Settings

NETWORK OBJECT	VALUE
Project network	<ul style="list-style-type: none"> Name: production-network1 Project: production
Project subnet	<ul style="list-style-type: none"> Name: production-subnet1 Network address: 192.168.1.0/24 DHCP Range: 192.168.1.2 to 192.168.1.254 DHCP: Enabled IP version: IPv4

- Create a new router for the **production** project.

Router Settings

TYPE	VALUE
Router name	production-router1
External network	provider-datacentre
Internal interface	production-subnet1 subnet

- Define a new security group, **default**. The security group must contain three rules, for SSH, ICMP, and HTTP access.

Security Group Settings

TYPE	VALUE
Security group name	default

TYPE	VALUE
Project	production
Security group rules	<ul style="list-style-type: none"> SSH traffic (port 22) from all networks HTTP traffic (port 80) from all networks ICMP traffic from all networks

- Define the new key pair **example-keypair1** in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect that file.
- Deploy an orchestration stack to launch an instance and verify the overcloud deployment.

Download the orchestration template from <http://materials.example.com/heat/review-template.yaml>. Launch the **review-stack** orchestration stack using an environment file.

Using the **ssh** command verify that the instance launched by the orchestration stack is accessible from **workstation**.

Evaluation

As the **student** user on **workstation**, run the **lab_compreview-s3** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab_compreview-s3 grade
```

Cleanup

From **workstation**, run the **lab_compreview-s3 cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab_compreview-s3 cleanup
```

This concludes the lab.

DEPLOYING OPENSTACK AND LAUNCHING A STACK

In this review, you will deploy OpenStack using director UI and launch an orchestration stack.

OUTCOMES

You should be able to:

- Create an overcloud plan directory with the director's base templates and include the plan environment file based on a particular deployment architecture.
- Import an overcloud plan using the web UI.
- Deploy the Red Hat OpenStack Platform overcloud using the director UI.
- Create resources for an orchestration stack.
- Launch an orchestration stack.

Confirm the **workstation** and the overcloud's virtual machines are started.

Set up your computers for this exercise by logging in to **workstation** as **student**, and run the following command:

```
[student@workstation ~]$ lab compreview-s3 setup
```

This command:

- Removes the **overcloud** deployment.
- Ensures the **controller0** and **compute0** bare metal nodes are configured.
- Deletes any existing overcloud plans.

Instructions

In this comprehensive review, you will import an overcloud plan into the Red Hat OpenStack Platform director UI and initiate the deployment using one controller and one compute node. Once the overcloud is deployed, you will use orchestration templates to deploy a stack that auto-scale a web server using the CPU utilization metrics.

The environment already provides the following resources for you:

- The Red Hat OpenStack Platform director UI is available at <https://director-ui.lab.example.com>. Use **admin** as user name and **redhat** as the password to access this web UI.
- The heat templates for director are available in **/usr/share/openstack-tripleo-heat-templates**.

- The **<http://materials.example.com/classroom-plan.tar.gz>** tar file contains a custom **plan-environment.yaml** file, and all the customization required for the Red Hat OpenStack Platform deployment in the classroom environment.
- The **<http://materials.example.com/osp-web.qcow2>** file contains an image of an installed RHEL system with a configured web server.

Configure your environment to meet the following requirements:

- Copy the **/usr/share/openstack-tripleo-heat-templates** directory on **director** in **/home/student/overcloud-poc** on **workstation**. In that directory, apply the customization provided by the **<http://materials.example.com/classroom-plan.tar.gz>** tar file.
- Update the custom **plan-environment.yaml** file in **/home/student/overcloud-poc** to force the overcloud **admin** user password to **redhat**. For that, add **AdminPassword: redhat** to the **parameter_defaults** section.
- Using the director web UI, import the overcloud plan provided by the **/home/student/overcloud-poc** directory. The director web UI is available at **<https://director-ui.lab.example.com>**. Use **admin** as user name and **redhat** as the password to access it.
- Ensure that one **Controller** role and one **Compute** role are assigned to the bare metal nodes.
- Deploy the overcloud. After the overcloud deployment process completes, the Red Hat OpenStack Platform dashboard for the overcloud should be available at **<http://dashboard.overcloud.example.com>**. Create the **admin** identity environment file in **/home/student/admin-rc** on workstation.
- Create the resources that you need later to deploy the autoscaling stack. First, create a flavor in the overcloud according to the following information.

Flavor Settings

TYPE	VALUE
Flavor name	default-no-swap
RAM	1024 MB
Root disk	10 GB
Number of VCPUs	2
Ephemeral disk	0
Swap	0

- Create a public image in the overcloud according to the following information.

Image Settings

TYPE	VALUE
Image name	rhel7-web
Disk image URL	http://materials.example.com/osp-web.qcow2
Protected	yes

TYPE	VALUE
Public	yes
Minimum disk size	10 GB
Minimum ram	1024 MB

- Create the **production** project in the overcloud.
- Create the **heat_stack_owner** role in the overcloud.
- Create a user for the **production** project according to the following information.

User Settings

TYPE	VALUE
User name	operator1
User project	production
User role	_member_, heat_stack_owner

- Create a privileged user for the **production** project.

Privileged User Settings

TYPE	VALUE
User name	architect1
User project	production
User role	admin

- Create a new cloud provider public network.

External Network Settings

NETWORK OBJECT	VALUE
Public network	<ul style="list-style-type: none"> • Name: provider-datacentre • External network: yes • Shared: yes • Provider network type: flat • Provider physical network: datacentre

NETWORK OBJECT	VALUE
Public subnet	<ul style="list-style-type: none"> Name: provider-subnet-172.25.250 Network address: 172.25.250.0/24 Range: 172.25.250.101 to 172.25.250.189 Gateway IP: 172.25.250.254 DNS server: 172.25.250.254 DHCP: Disabled IP version: IPv4

- Create a new network for the **production** project.

Project Network Settings

NETWORK OBJECT	VALUE
Project network	<ul style="list-style-type: none"> Name: production-network1 Project: production
Project subnet	<ul style="list-style-type: none"> Name: production-subnet1 Network address: 192.168.1.0/24 DHCP Range: 192.168.1.2 to 192.168.1.254 DHCP: Enabled IP version: IPv4

- Create a new router for the **production** project.

Router Settings

TYPE	VALUE
Router name	production-router1
External network	provider-datacentre
Internal interface	production-subnet1 subnet

- Define a new security group, **default**. The security group must contain three rules, for SSH, ICMP, and HTTP access.

Security Group Settings

TYPE	VALUE
Security group name	default

TYPE	VALUE
Project	production
Security group rules	<ul style="list-style-type: none"> SSH traffic (port 22) from all networks HTTP traffic (port 80) from all networks ICMP traffic from all networks

- Define the new key pair **example-keypair1** in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect that file.
- Deploy an orchestration stack to launch an instance and verify the overcloud deployment.

Download the orchestration template from <http://materials.example.com/heat/review-template.yaml>. Launch the **review-stack** orchestration stack using an environment file.

Using the **ssh** command verify that the instance launched by the orchestration stack is accessible from **workstation**.

- Copy the **/usr/share/openstack-tripleo-heat-templates** directory on **director** in **/home/student/overcloud-poc** on **workstation**. In that directory, apply the customization provided by the <http://materials.example.com/classroom-plan.tar.gz> tar file.
 - Copy the **/usr/share/openstack-tripleo-heat-templates** directory on **director** in **/home/student/overcloud-poc** on **workstation**.

```
[student@workstation ~]$ scp -r \
stack@director:/usr/share/openstack-tripleo-heat-templates \
/home/student/overcloud-poc
...output omitted...
```

- Download and extract <http://materials.example.com/classroom-plan.tar.gz> in **/home/student/overcloud-poc**.

```
[student@workstation ~]$ cd /home/student/overcloud-poc
[student@workstation overcloud-poc]$ wget \
http://materials.example.com/classroom-plan.tar.gz
...output omitted...
[student@workstation overcloud-poc]$ tar xvf classroom-plan.tar.gz
...output omitted...
```

- Update the custom **plan-environment.yaml** file in **/home/student/overcloud-poc** to force the overcloud **admin** user password to **redhat**. For that, add **AdminPassword: redhat** to the **parameter_defaults** section.

```
[student@workstation overcloud-poc]$ vim plan-environment.yaml
...output omitted...
parameter_defaults:
  AdminPassword: redhat
  NodeRootPassword: redhat
  OvercloudComputeFlavor: baremetal
  OvercloudControllerFlavor: baremetal
```



WARNING

Be careful with indentation. You can only use space characters; do not use tab characters. Syntax errors may lead to the failure of the overcloud deployment.

3. Using the director web UI, import the overcloud plan provided by the **/home/student/overcloud-poc** directory. The director web UI is available at **<https://director-ui.lab.example.com>**. Use **admin** as user name and **redhat** as the password to access it.
 - 3.1. Open the **<https://director-ui.lab.example.com>** URL to access the director web UI. If a certificate error appears, accept the self-signed certificate. Log in to the director UI using **admin** as user name and **redhat** as the password.
 - 3.2. From the Red Hat OpenStack Platform director dashboard click Import Plan in the Plan tab.
 - 3.3. In the New Plan tab, enter **overcloud-poc** as the Plan Name.
 - 3.4. Select Local Folder as the Upload Type.
 - 3.5. Click Browse... to browse and select the **overcloud-poc** directory in the **student** home directory.
 - 3.6. Click Upload Files and Create Plan. Wait till the overcloud plan files are uploaded and the plan is created on the director node.
4. Ensure that one **Controller** role and one **Compute** role are assigned to the bare metal nodes.
 - 4.1. From the Plans tab, click the **overcloud-poc** plan card. The web page shows four steps to deploy the **overcloud-poc** plan.
 - 4.2. In the Configure Roles and Assign Nodes section, ensure that two nodes are selected. One is assigned the **Controller** role and the other is assigned the **Compute** role. You may have to wait a few minutes for the interface to display the assignments.
5. Deploy the overcloud. After the overcloud deployment process completes, create the **admin** identity environment file in **/home/student/admin-rc** on workstation.
 - 5.1. Click Validate and Deploy to start the deployment validation.



NOTE

If any of the pre-deployment validations fails, a warning message appears. For the classroom deployment, this is expected.

- 5.2. Click Deploy to start the overcloud deployment.
- 5.3. After the overcloud deployment process completes, the Deploy section displays the overcloud IP address, user name, and password to access the overcloud dashboard.

Overcloud information:

Overcloud IP address: 172.25.250.50
Username: admin
Password: redhat

- 5.4. Use the overcloud Red Hat OpenStack Platform dashboard to download the **admin** identity environment file. In the browser, log in to **<http://>**

dashboard.overcloud.example.com using the **admin** user and **redhat** for the password.

- 5.5. Click the Project tab from the top. Navigate to Project → API Access. On the **API Access** page, navigate to Download OpenStack RC File → OpenStack RC File (Identity API v3). Save the environment file and log out from the dashboard.
- 5.6. Copy the downloaded file to **/home/student/admin-rc** and edit the file.

```
[student@workstation overcloud-poc]$ cd  
[student@workstation ~]$ cp ~/Downloads/admin-openrc.sh ~/admin-rc  
[student@workstation ~]$ vim ~/admin-rc  
...output omitted...  
export OS_AUTH_URL=http://172.25.250.50:5000/v3  
# With the addition of Keystone we have standardized on the term **project**  
# as the entity that owns the resources.  
#export OS_PROJECT_ID=9e5bda8d6661470ca2fce55326b0f50a  
export OS_PROJECT_NAME="admin"  
export OS_USER_DOMAIN_NAME="Default"  
...output omitted...  
export OS_USERNAME="admin"  
# With Keystone you pass the keystone password.  
# echo "Please enter your OpenStack Password for project $OS_PROJECT_NAME as user  
$OS_USERNAME: "  
# read -sr OS_PASSWORD_INPUT  
export OS_PASSWORD=redhat  
...output omitted...
```

In that file you have to comment out the **OS_PROJECT_ID** variable and the two lines that interactively ask for the password, and define the password in the **OS_PASSWORD** variable.

6. Create the **default-no-swap** flavor in the overcloud.

- 6.1. Source the identity environment file for the **admin** administrative account.

```
[student@workstation ~]$ source ~/admin-rc
```

- 6.2. Create the **default-no-swap** flavor.

```
[student@workstation ~]$ openstack flavor create default-no-swap \  
--ram 1024 --vcpus 2 --disk 10 --swap 0 --ephemeral 0  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| OS-FLV-DISABLED:disabled | False |  
| OS-FLV-EXT-DATA:ephemeral | 0 |  
| disk | 10 |  
| id | 87d1f723-b264-4f56-bc6c-04aca241ca99 |  
| name | default-no-swap |  
| os-flavor-access:is_public | True |  
| properties | |  
| ram | 1024 |  
| rxtx_factor | 1.0 |  
| swap | |  
| vcpus | 2 |  
+-----+-----+
```

7. Create the **rhel7-web** public image from the <http://materials.example.com/osp-web.qcow2> disk image.

7.1. Download the image.

```
[student@workstation ~]$ wget http://materials.example.com/osp-web.qcow2
...output omitted...
```

7.2. Create the **rhel7-web** image.

```
[student@workstation ~]$ openstack image create \
--protected \
--public \
--min-disk 10 \
--min-ram 1024 \
--disk-format qcow2 \
--file osp-web.qcow2 \
rhel7-web
+-----+
| Field      | Value
+-----+
| checksum   | 0bd99215aa4360c02af6a291f0fa6b39e
| container_format | bare
| created_at | 2018-06-26T06:27:35Z
| disk_format | qcow2
| file       | /v2/images/0860d85f-29cb-4b64-94e6-598b9f620a4f/file
| id         | 0860d85f-29cb-4b64-94e6-598b9f620a4f
| min_disk   | 10
| min_ram   | 1024
| name       | rhel7-web
| owner      | 277a119da27344809451b271e3f7a5db
| properties | direct_url='...'
| protected  | True
| schema     | /v2/schemas/image
| size       | 971636736
| status     | active
| tags       |
| updated_at | 2018-06-26T06:27:47Z
| virtual_size | None
| visibility | public
+-----+
```

8. Create the **production** project in the overcloud.

```
[student@workstation ~]$ openstack project create production
+-----+
| Field      | Value
+-----+
| description | 
| domain_id  | default
| enabled    | True
| id         | dba264c627b84094b468d19714fbab61
| is_domain  | False
| name       | production
| parent_id  | default
| tags       | []
```

9. Create the **heat_stack_owner** role in the overcloud.

```
[student@workstation ~]$ openstack role create heat_stack_owner
+-----+-----+
| Field      | Value
+-----+-----+
| domain_id  | default
| id         | cfgfd64c627b84094b468d19714fb413
| name       | heat_stack_owner
+-----+
```

10. Create a the **operator1** user for the **production** project.

- 10.1. Create the **operator1** user, and give the user the **_member_**, and **heat_stack_owner** role in the **production** project.

```
[student@workstation ~]$ openstack user create --project production \
--password redhat operator1
+-----+-----+
| Field      | Value
+-----+-----+
| default_project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da
| domain_id    | default
| enabled      | True
| id           | 7d063fb0138349948a2611956bafb868
| name         | operator1
| options      | {}
| password_expires_at | None
+-----+
[student@workstation ~]$ openstack role add --project production \
--user operator1 _member_
[student@workstation ~]$ openstack role add --project production \
--user operator1 heat_stack_owner
```

- 10.2. Create the identity environment file for the new user by copying the **/home/student/admin-rc** file to **/home/student/operator1-production-rc** and updating that file.

```
[student@workstation ~]$ cp ~/admin-rc ~/operator1-production-rc
[student@workstation ~]$ vim ~/operator1-production-rc
export CLOUDPROMPT_ENABLED=0
export OS_NO_CACHE=True
export COMPUTE_API_VERSION=1.1

export OS_USERNAME=operator1

export no_proxy=,172.25.250.50,172.25.249.50
export OS_USER_DOMAIN_NAME=Default
export OS_VOLUME_API_VERSION=3
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://172.25.250.50:5000//v3
export NOVA_VERSION=1.1
export OS_IMAGE_API_VERSION=2
export OS_PASSWORD=redhat
```

```

export OS_PROJECT_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3

export OS_PROJECT_NAME=production

export OS_AUTH_TYPE=password
export PYTHONWARNINGS="ignore:Certificate has no, ignore:A true SSLContext object
is not available"

export PS1='[\u@h \w(operator1-production)]\$ '

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
    export PS1=${PS1:+$"}
    export PS1=\${OS_CLOUDNAME:+"\${OS_CLOUDNAME}"}\ $PS1
    export CLOUDPROMPT_ENABLED=1
fi

```



NOTE

Pay attention to the parameters mentioned in bold in the above screen.

11. Create the **architect1** privileged user in the **production** project.

- 11.1. Create the **architect1** user, and give the user the **admin** role in the **production** project.

```

[student@workstation ~]$ openstack user create --project production \
--password redhat architect1
+-----+-----+
| Field          | Value           |
+-----+-----+
| default_project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| domain_id      | default         |
| enabled         | True            |
| id              | aa3b2365a3b947ef91b37aba64b31d91 |
| name            | architect1      |
| options          | {}              |
| password_expires_at | None            |
+-----+-----+
[student@workstation ~]$ openstack role add --project production \
--user architect1 admin

```

- 11.2. Create the identity environment file for the new user by copying the **/home/student/admin-rc** file to **/home/student/architect1-production-rc** and updating that file.

```

[student@workstation ~]$ cp ~/admin-rc ~/architect1-production-rc
[student@workstation ~]$ vim ~/architect1-production-rc
export CLOUDPROMPT_ENABLED=0
export OS_NO_CACHE=True
export COMPUTE_API_VERSION=1.1

export OS_USERNAME=architect1

export no_proxy=,172.25.250.50,172.25.249.50

```

```

export OS_USER_DOMAIN_NAME=Default
export OS_VOLUME_API_VERSION=3
export OS_CLOUDNAME=overcloud
export OS_AUTH_URL=http://172.25.250.50:5000//v3
export NOVA_VERSION=1.1
export OS_IMAGE_API_VERSION=2
export OS_PASSWORD=redhat
export OS_PROJECT_DOMAIN_NAME=Default
export OS_IDENTITY_API_VERSION=3

export OS_PROJECT_NAME=production

export OS_AUTH_TYPE=password
export PYTHONWARNINGS="ignore:Certificate has no, ignore:A true SSLContext object
is not available"

export PS1='[\u@ \h \W(\architect1-production)]\$ '

# Add OS_CLOUDNAME to PS1
if [ -z "${CLOUDPROMPT_ENABLED:-}" ]; then
    export PS1=${PS1:-"}
    export PS1=\${OS_CLOUDNAME:+"\($OS_CLOUDNAME\)"}\ $PS1
    export CLOUDPROMPT_ENABLED=1
fi

```



NOTE

Pay attention to the parameters mentioned in bold in the above screen.

12. Create the public **provider-datacentre** provider network and the **provider-subnet-172.25.250** subnet.
 - 12.1. Using the **admin** environment file, create the public **provider-datacentre** network.

```
[student@workstation ~]$ openstack network create \
--external \
--share \
--provider-network-type flat \
--provider-physical-network datacentre \
provider-datacentre
+-----+-----+
| Field          | Value        |
+-----+-----+
| admin_state_up | UP           |
| availability_zone_hints | |
| availability_zones | |
| created_at     | 2018-06-21T10:47:44Z |
| description     | |
| dns_domain     | None          |
| id             | 2256afc6-4c24-4942-b419-1d975d06d18a |
| ipv4_address_scope | None          |
| ipv6_address_scope | None          |
| is_default      | False          |
| is_vlan_transparent | None          |
| mtu             | 1500          |
```

name	provider-datacentre
port_security_enabled	True
project_id	50f29df5b0144794a4db5dad1eb12d9a
provider:network_type	flat
provider:physical_network	datacentre
provider:segmentation_id	None
qos_policy_id	None
revision_number	6
router:external	External
segments	None
shared	True
status	ACTIVE
subnets	
tags	
updated_at	2018-06-21T10:47:45Z

12.2. Create the **provider-subnet-172.25.250** subnet.

[student@workstation ~]\$ openstack subnet create \
--no-dhcp \
--subnet-range 172.25.250.0/24 \
--gateway 172.25.250.254 \
--dns-nameserver 172.25.250.254 \
--allocation-pool start=172.25.250.101,end=172.25.250.189 \
--network provider-datacentre \
provider-subnet-172.25.250
+-----+-----+-----+
Field Value
+-----+-----+-----+
allocation_pools 172.25.250.101-172.25.250.189
cidr 172.25.250.0/24
created_at 2018-06-21T10:52:51Z
description
dns_nameservers 172.25.250.254
enable_dhcp False
gateway_ip 172.25.250.254
host_routes
id 3cfac2f2-a200-4264-acfb-28f0e96d2c3d
ip_version 4
ipv6_address_mode None
ipv6_ra_mode None
name provider-subnet-172.25.250
network_id 2256afc6-4c24-4942-b419-1d975d06d18a
project_id 50f29df5b0144794a4db5dad1eb12d9a
revision_number 0
segment_id None
service_types
subnetpool_id None
tags
updated_at 2018-06-21T10:52:51Z

13. Create the project **production-network1** network and the **production-subnet1** subnet for the **production** project.

- 13.1. Source the identity environment file for the **operator1** account and create the **production-network1** network.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack network create \
production-network1
+-----+-----+
| Field | Value |
+-----+-----+
| admin_state_up | UP |
| availability_zone_hints | |
| availability_zones | |
| created_at | 2018-06-21T10:37:57Z |
| description | |
| dns_domain | None |
| id | 7fdc48c7-6f12-4f00-8e70-b0db418ec8ee |
| ipv4_address_scope | None |
| ipv6_address_scope | None |
| is_default | False |
| is_vlan_transparent | None |
| mtu | 1450 |
| name | production-network1 |
| port_security_enabled | True |
| project_id | 83aaaf87d7d0b4f49a27b7c7e597d26da |
| provider:network_type | None |
| provider:physical_network | None |
| provider:segmentation_id | None |
| qos_policy_id | None |
| revision_number | 3 |
| router:external | Internal |
| segments | None |
| shared | False |
| status | ACTIVE |
| subnets | |
| tags | |
| updated_at | 2018-06-21T10:37:57Z |
+-----+-----+
```

- 13.2. Create the **production-subnet1** subnet.

```
[student@workstation ~(operator1-production)]$ openstack subnet create \
--subnet-range 192.168.1.0/24 \
--dhcp \
--allocation-pool start=192.168.1.2,end=192.168.1.254 \
--network production-network1 \
production-subnet1
+-----+-----+
| Field | Value |
+-----+-----+
| allocation_pools | 192.168.1.2-192.168.1.254 |
| cidr | 192.168.1.0/24 |
| created_at | 2018-06-21T10:42:00Z |
| description | |
```

dns_nameservers	
enable_dhcp	True
gateway_ip	192.168.1.1
host_routes	
id	754c622c-6258-4e88-940f-0ae30c39a1a2
ip_version	4
ipv6_address_mode	None
ipv6_ra_mode	None
name	production-subnet1
network_id	7fdc48c7-6f12-4f00-8e70-b0db418ec8ee
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
revision_number	0
segment_id	None
service_types	
subnetpool_id	None
tags	
updated_at	2018-06-21T10:42:00Z

14. Create the **production-router1** router in the **production** project. Connect it to the **production-subnet1** private subnet and define it as a gateway for the **provider-datacentre** external network.

- 14.1. Source the identity environment file for the **operator1** account and create the **production-router1** router.

[student@workstation ~]\$ source ~/operator1-production-rc	
[student@workstation ~(operator1-production)]\$ openstack router create \	
production-router1	
+-----+-----+	
Field	Value
+-----+-----+	
admin_state_up	UP
availability_zone_hints	
availability_zones	
created_at	2018-06-21T11:36:01Z
description	
distributed	False
external_gateway_info	None
flavor_id	None
ha	False
id	74cc9507-f70a-4e39-bd6c-1f3289e54b33
name	production-router1
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
revision_number	1
routes	
status	ACTIVE
tags	
updated_at	2018-06-21T11:36:01Z
+-----+-----+	

- 14.2. Connect the **production-router1** router to the **production-subnet1** private subnet. Define it as a gateway for the **provider-datacentre** external network.

```
[student@workstation ~(operator1-production)]$ openstack router add subnet \
production-router1 production-subnet1
```

```
[student@workstation ~] $ openstack router set \
--external-gateway provider-datacentre production-router1
```

15. Define the **default** security group in the **production** project. Add rules for SSH, ICMP, and HTTP.

- 15.1. Create the **default** security group.

```
[student@workstation ~] $ openstack security group \
create default
Error while executing command: HttpException: Unknown error, {"NeutronError": \
{"message": "Default security group already exists.", "type": \
"SecurityGroupDefaultAlreadyExists", "detail": ""}}
```

Notice that the **default** security group already exists because all the projects get a default security group, named **default**, at creation time.

- 15.2. Add the rules to the security group:

- SSH traffic (port 22) from all networks
- HTTP traffic (port 80) from all networks
- ICMP traffic from all networks

```
[student@workstation ~] $ openstack security group rule \
create --proto tcp --dst-port 22:22 default
```

Field	Value
created_at	2018-06-21T11:49:45Z
description	
direction	ingress
ether_type	IPv4
id	9e955c89-3835-4170-a61e-2e50d0176b20
name	None
port_range_max	22
port_range_min	22
project_id	83aaaf87d7d0b4f49a27b7c7e597d26da
protocol	tcp
remote_group_id	None
remote_ip_prefix	0.0.0.0/0
revision_number	0
security_group_id	694e9d59-1c53-40a8-82bf-11c5c169460d
updated_at	2018-06-21T11:49:45Z

```
[student@workstation ~] $ openstack security group rule \
create --proto tcp --dst-port 80:80 default
```

Field	Value
created_at	2018-06-21T11:50:55Z
description	
direction	ingress
ether_type	IPv4
id	792b9f3a-334c-4c1b-8cad-2ae77e425454
name	None

```

| port_range_max | 80
| port_range_min | 80
| project_id     | 83aaaf87d7d0b4f49a27b7c7e597d26da
| protocol       | tcp
| remote_group_id | None
| remote_ip_prefix | 0.0.0.0/0
| revision_number | 0
| security_group_id | 694e9d59-1c53-40a8-82bf-11c5c169460d
| updated_at      | 2018-06-21T11:50:55Z
+-----+
[student@workstation ~-(operator1-production)]$ openstack security group rule \
create --proto icmp default
+-----+-----+
| Field        | Value
+-----+-----+
| created_at   | 2018-06-21T11:51:57Z
| description   |
| direction     | ingress
| ether_type    | IPv4
| id            | c5aa51d9-70b3-4c72-a7a6-95fe1997d433
| name          | None
| port_range_max | None
| port_range_min | None
| project_id    | 83aaaf87d7d0b4f49a27b7c7e597d26da
| protocol      | icmp
| remote_group_id | None
| remote_ip_prefix | 0.0.0.0/0
| revision_number | 0
| security_group_id | 694e9d59-1c53-40a8-82bf-11c5c169460d
| updated_at    | 2018-06-21T11:51:57Z
+-----+

```

16. Create the **example-keypair1** key pair in the **production** project. Save the private key on **workstation** in **/home/student/.ssh/example-keypair1** and protect it.

```

[student@workstation ~-(operator1-production)]$ openstack keypair create \
example-keypair1 > ~/.ssh/example-keypair1
[student@workstation ~-(operator1-production)]$ chmod 600 ~/.ssh/example-keypair1

```

17. Deploy an orchestration stack to launch an instance and verify the overcloud deployment. Download the orchestration template from <http://materials.example.com/heat/review-template.yaml>. Launch the **review-stack** orchestration stack using an environment file.

Using the **ssh** command verify that the instance launched by the orchestration stack is accessible from **workstation**.

- 17.1. Download the orchestration template from <http://materials.example.com/heat/review-template.yaml>.

```

[student@workstation ~-(operator1-production)]$ wget \
http://materials.example.com/heat/review-template.yaml

```

...output omitted...

- 17.2. Review the **review-template.yaml** file and create an environment file, **/home/student/review-environment.yaml**.

```
parameters:  
  keypair: "example-keypair1"  
  image: "rhel7-web"  
  flavor: "default-no-swap"  
  instance_name: "production-server1"  
  external_network: "provider-datacentre"  
  internal_network: "production-network1"  
  internal_subnet: "production-subnet1"
```

- 17.3. Launch the orchestration stack with the name as **review-stack**.

```
[student@workstation ~ (operator1-production)]$ openstack stack create \  
--environment ~/review-environment.yaml \  
--template ~/review-template.yaml \  
--wait review-stack  
...output omitted...  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| id | 85407ca4-893a-4e83-b902-59373f58d7f7 |  
| stack_name | review-stack |  
| description | launching a custom instance |  
| creation_time | 2018-06-28T08:34:30Z |  
| updated_time | None |  
| stack_status | CREATE_COMPLETE |  
| stack_status_reason | Stack CREATE completed successfully |  
+-----+-----+
```

- 17.4. View the output parameters and their values for the **review-stack** stack.

```
[student@workstation ~ (operator1-production)]$ openstack stack output list \  
review-stack  
+-----+-----+  
| output_key | description |  
+-----+-----+  
| instance_private_ip | Private IP address of the instance |  
| instance_public_ip | Public IP address of the instance |  
+-----+-----+
```

View the value for the *instance_public_ip* output parameter.

```
[student@workstation ~ (developer1-finance)]$ openstack stack output show \  
review-stack instance_public_ip  
+-----+-----+  
| Field | Value |  
+-----+-----+  
| description | Public IP address of the instance |  
| output_key | instance_public_ip |  
| output_value | 172.25.250.P |
```

- 17.5. List the instances present in the environment to check the status of the **production-server1** instance.

```
[student@workstation ~](operator1-production)]$ openstack server list \
-c Name -c Status
+-----+-----+
| Name | Status |
+-----+-----+
| production-server1 | ACTIVE |
+-----+
```

- 17.6. Verify that the **production-server1** is accessible from **workstation**.

```
[student@workstation ~](operator1-production)]$ ssh \
-i ~/ssh/example-keypair1 cloud-user@172.25.250.P
[cloud-user@production-server1 ~]$
```

Log out of the instance.

```
[cloud-user@prduction-server1 ~]$ exit
[student@workstation ~](operator1-production)]$
```

Evaluation

As the **student** user on **workstation**, run the **lab_compreview-s3** script with the **grade** argument to confirm success on this exercise. Correct any reported failures and rerun the script until successful.

```
[student@workstation ~]$ lab_compreview-s3 grade
```

Cleanup

From **workstation**, run the **lab_compreview-s3 cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab_compreview-s3 cleanup
```

This concludes the lab.