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 (ROLE)

Red Hat OpenStack Platform 13.0 CL210

RED HAT OPENSTACK ADMINISTRATION II: INFRASTRUCTURE CONFIGURATION FOR CLOUD ADMINISTRATORS

Edition 1

redhat.

RED HAT OPENSTACK ADMINISTRATION II: INFRASTRUCTURE CONFIGURATION FOR CLOUD ADMINISTRATORS

redhat

Red Hat OpenStack Platform 13.0 CL210
Red Hat OpenStack Administration II: Infrastructure Configuration for Cloud Administrators
Edition 1 20181129
Publication date 20181129

Authors: Chen Chang, Fiona Allen, Morgan Weetman, Saumik Paul, Snehangshu Karmakar
Editor: Philip Sweany, David O'Brien

Copyright © 2017 Red Hat, Inc.

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

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

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

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

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

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

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

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

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

All other trademarks are the property of their respective owners.

Document Conventions vii

Introduction ix
Red Hat OpenStack Administration II: Infrastructure Configuration for Cloud Administrators ... ix
Orientation to the Classroom Environment ... x
Controlling Your Systems ... xvii
Internationalization ... xix

1. Navigating the Red Hat OpenStack Platform Architecture 1
Introducing the Red Hat OpenStack Platform Infrastructure ... 2
Guided Exercise: Navigating the Red Hat OpenStack Platform Infrastructure ... 9
Describing the Undercloud ... 14
Guided Exercise: Viewing the Undercloud Architecture ... 23
Introducing Containerized Services ... 27
Guided Exercise: Viewing Containerized Service Structures ... 37
Describing the Overcloud ... 42
Guided Exercise: Viewing the Overcloud Architecture ... 56
Lab: Navigating the Red Hat OpenStack Platform Architecture ... 64
Summary ... 75

2. Describing the OpenStack Control Plane 77
Describing Service Access by API ... 78
Quiz: Describing Service Access by API ... 83
Describing Service Communication by Message Broker ... 85
Guided Exercise: Describing Service Communication by Message Broker ... 99
Identifying Overcloud Control Plane Services ... 104
Guided Exercise: Viewing Overcloud Control Plane Services ... 113
Lab: Managing Internal OpenStack Communication ... 121
Summary ... 127

3. Integrating Identity Management 129
Managing an Integrated IdM Back-end Configuration ... 130
Guided Exercise: Verifying an IdM Back-end Configuration ... 138
Managing Identity Service Tokens ... 141
Guided Exercise: Managing Identity Service Tokens ... 147
Managing Project Organization ... 151
Guided Exercise: Managing Project Organization ... 158
Lab: Integrating Identity Management ... 161
Summary ... 167

4. Performing Image Operations 169
Comparing Image Formats ... 170
Quiz: Describing Image Formats ... 174
Building Customized Images ... 176
Guided Exercise: Building an Image ... 181
Modifying Guest and Disk Images ... 186
Guided Exercise: Customizing an Image ... 189
Initializing an Instance During Deployment ... 196
Guided Exercise: Initializing an Instance During Deployment ... 200
Lab: Performing Image Operations ... 203
Summary ... 210

5. Managing Storage 211
Implementing Block Storage ... 212
Guided Exercise: Implementing Block Storage ... 225
Comparing Object Stores ... 231
Guided Exercise: Comparing Object Stores ... 239
Managing Shared File Systems ... 245

Guided Exercise: Managing Shared File Systems 250
Managing Ephemeral and Persistent Storage 257
Guided Exercise: Managing Ephemeral and Persistent Storage 261
Lab: Managing Storage 270
Summary 287

6. Managing OpenStack Networking **289**
Describing Network Protocol Types 290
Guided Exercise: Describing Network Protocol Types 300
Introducing Open Virtual Networking (OVN) 310
Guided Exercise: Implementing Open Virtual Networks (OVN) 323
Selecting Network Provisioning Options 334
Guided Exercise: Selecting Network Provisioning Options 339
Lab: Managing OpenStack Networking 343
Summary 353

7. Managing Compute Resources **355**
Describing the Instance Launch Process 356
Guided Exercise: Describing the Instance Launch Process 361
Introducing Red Hat Hyperconverged Infrastructure 370
Guided Exercise: Launching to a Hyperconverged Compute Node 375
Administering Compute Nodes 382
Guided Exercise: Administering Compute Nodes 387
Lab: Managing Compute Resources 391
Summary 401

8. Automating Cloud Applications **403**
Creating Application Stacks 404
Quiz: Creating Application Stacks 409
Writing Heat Orchestration Templates 411
Guided Exercise: Writing Heat Orchestration Templates 422
Lab: Automating Cloud Applications 428
Summary 435

9. Troubleshooting OpenStack Operations **437**
Diagnosing OpenStack Issues 438
Guided Exercise: Diagnosing OpenStack Issues 449
Troubleshooting Common Core Issues 457
Guided Exercise: Troubleshooting Common Core Issues 469
Lab: Troubleshooting OpenStack 481
Summary 492

10. Comprehensive Review **493**
Comprehensive Review 494
Lab: Building a Custom Image 496
Lab: Managing Project Organization 508
Lab: Launching an Application Stack 516

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 II: INFRASTRUCTURE CONFIGURATION FOR CLOUD ADMINISTRATORS

Red Hat OpenStack Administration II (CL210) is designed for system administrators who intend to implement a cloud computing environment using OpenStack. Students will learn how to configure, use, and maintain Red Hat OpenStack Platform.

The focus of this course is managing OpenStack using the unified command-line interface, managing instances, and maintaining an enterprise deployment of OpenStack. Exam competencies covered in the course include: manage projects, images, networking, object storage, and block storage; provide orchestration; and build a customized image.

COURSE OBJECTIVES

- Design and implement on-demand projects, software-defined networks, and virtual machine instances.
- Troubleshoot individual services as well as OpenStack holistically.
- Manage the migration of live instances.
- Create and customize advanced VM instances as applications: customize on deploy, create scalable stacks.

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 Red Hat Enterprise Linux) certification or equivalent experience.
- Red Hat OpenStack Administration I (CL110) course or equivalent experience.

ORIENTATION TO THE CLASSROOM ENVIRONMENT

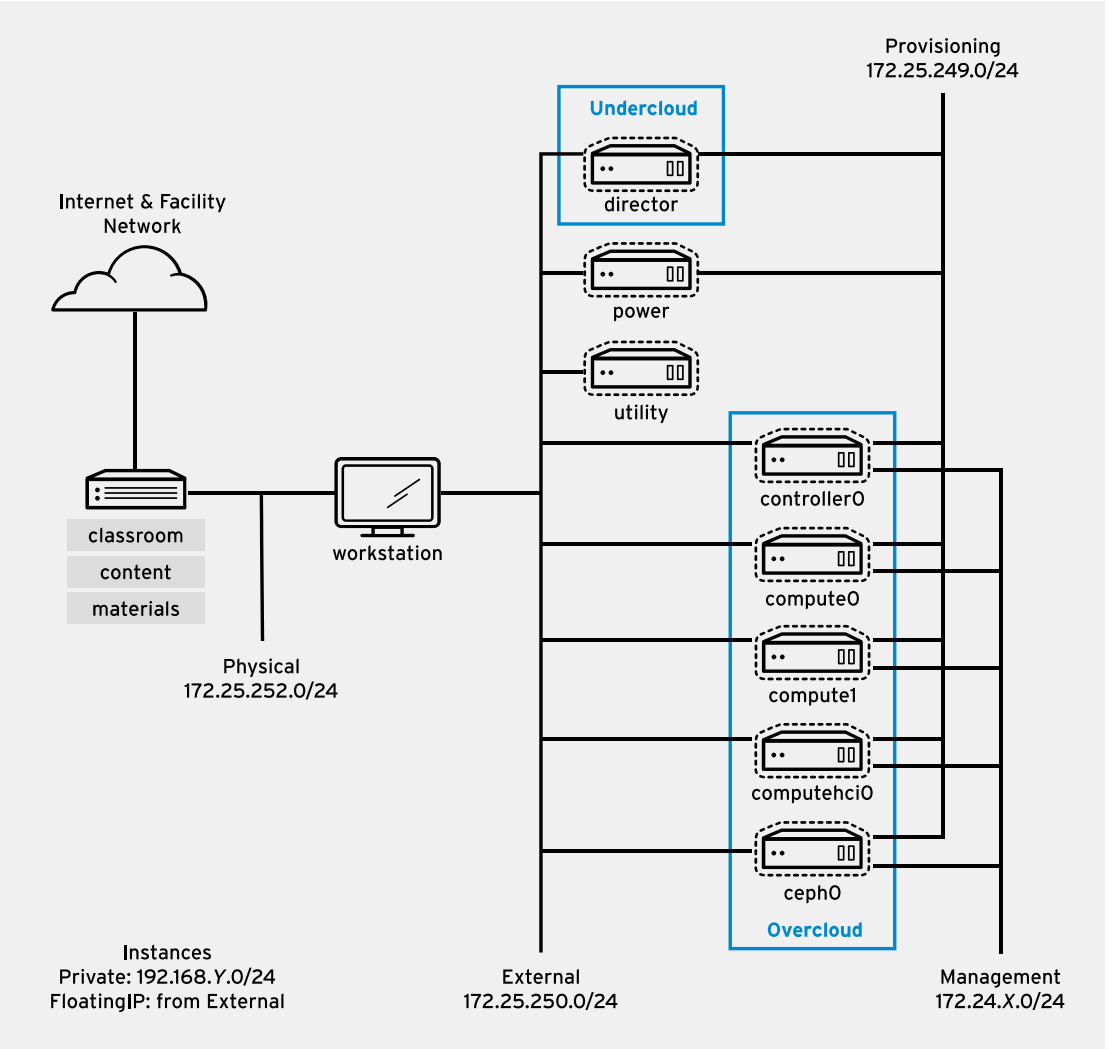


Figure 0.1: Classroom environment

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 `utility`, `director`, and `power` in the `lab.example.com` DNS domain, and `controller0`, `compute0`, `compute1`, `computehci0`, 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, **workstation*N*.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	172.25.250.100 172.25.249.100 172.25.249.101 172.25.249.102 172.25.249.112 172.25.249.103 172.25.249.106	Handles overcloud node IPMI power management
utility.lab.example.com	172.25.250.220 172.24.250.220	Identity management server and VLANS for provider networks
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
compute1.overcloud.example.com	172.25.250.12 172.25.249.*S* 172.24.*X*.12	Another overcloud compute node
computehci0.overcloud.example.com	172.25.250.6 172.25.249.*T* 172.24.*X*.6	An overcloud compute node
ceph0.overcloud.example.com	172.25.250.3 172.25.249.*U* 172.24.*X*.3	The overcloud block and object storage node
classroom.example.com	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 CREDENTIALS	USERNAME	PASSWORD
Red Hat Identity Manager admin	`admin`	`RedHat123^`
Red Hat OpenStack Platform dashboard admin	`admin`	`redhat`
Red Hat OpenStack Platform dashboard user	*as directed*	`redhat`
Red Hat OpenStack Platform director	`admin`	`redhat`

MANAGING OPENSTACK OVERCLOUDS IN THE CLASSROOM

Classroom OpenStack overclouds require continuous communication and coordination between the undercloud director node, the power node, and overcloud nodes. *This discussion focuses on the classroom procedures only*, because production OpenStack installations are highly available, redundant, resilient, and scalable infrastructures that *rarely 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 manage their overcloud using the procedures described here.

Figure 0.2 shows the infrastructure design. Classroom components, networks, IP addresses, and interface configuration will be referred to throughout this course, and will be discussed or explained in upcoming chapters. The configuration shown here was pre-built using RHOSP Director using customized TripleO templates. Network interfaces shown here without a connecting network or IP addresses (e.g., `eth3` and `eth4` on many systems) are extra interfaces used in an upcoming chapter to configure provider networks.

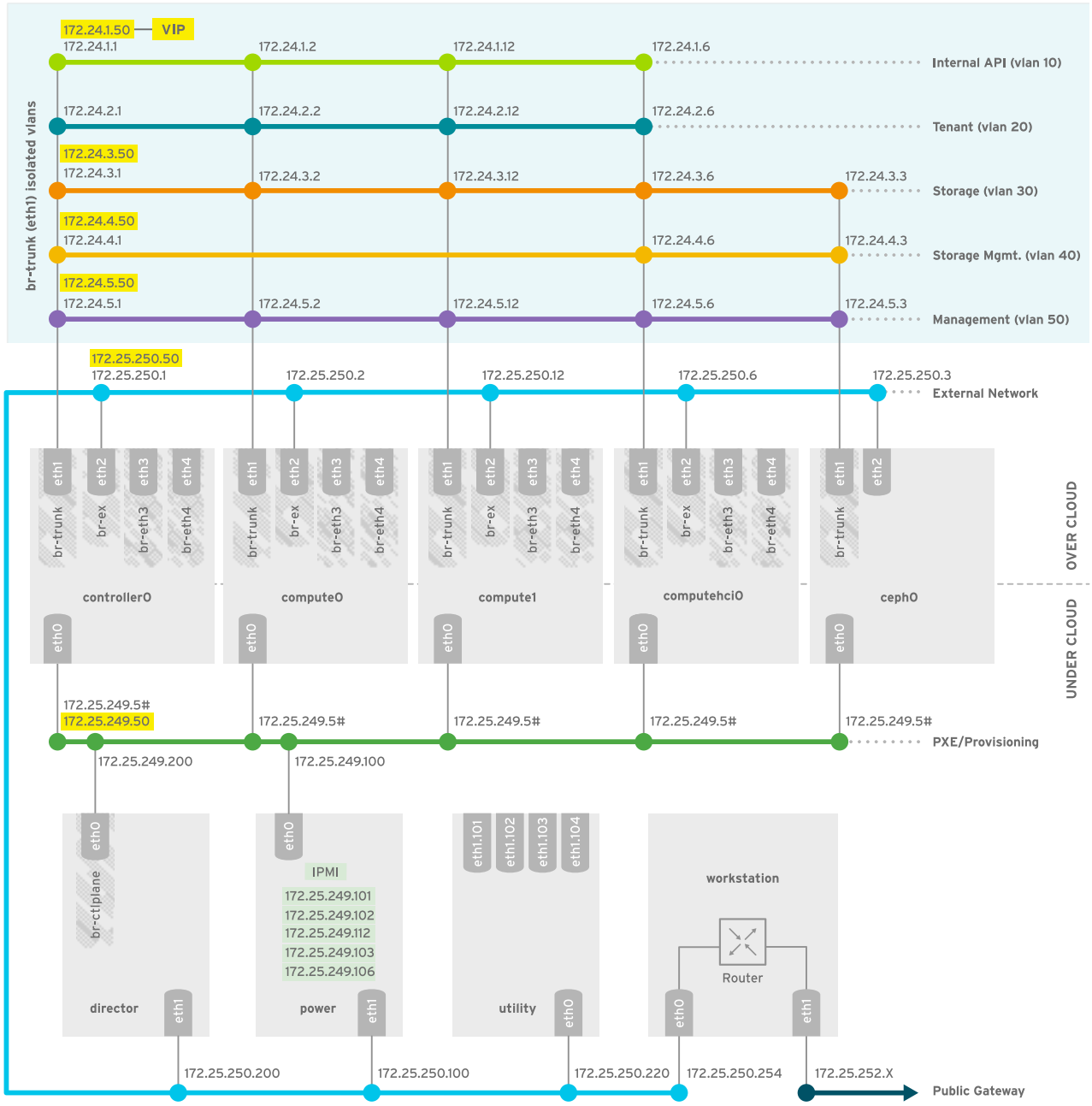


Figure 0.2: CL210-RHOSP13 classroom network topology

Shutting Down the Classroom OpenStack Cluster

The procedure described here is only for use in the classroom environment, because it shuts down the overcloud VMs using an ungraceful, forced technique. The procedure for gracefully shutting down a production overcloud requires stopping services in a logical, dependent order. That proper production procedure is taught in an upcoming chapter in this course.

Unlike a production overcloud, this classroom environment is not intended to remain running continuously throughout the course. The online (ROL) classroom provider automatically shuts down VMs after a user-set timer expiration. Similarly, the physical (ILT) classroom providers typically turn off physical machines overnight. Because proper shutdown procedures are time-consuming in a classroom, we allow less graceful procedures that are not acceptable in a production environment.

> **CAUTION**
> Stopping overcloud nodes without properly stopping overcloud services first could cause data loss or corruption, and is not a recommended practice. However, since current versions of OpenStack are resilient, using shortcut shutdown procedures rarely causes problems in the classroom environment. Always use the recommended, supported procedures in production.

In the online classroom environment, all classroom VMs shut down when the user-set timer expires. It is not uncommon to forget that the timer is set, or to leave for the day without shutting down. In a physical classroom, shutting down the physical machine without taking any preliminary actions will have the same result. In these cases, the overcloud will be stopped ungracefully. Although not recommended, this forced shutdown rarely causes problems. At next startup, the overcloud typically returns to a normal and working state.

However, students must still remember to delete instances in the overcloud. Deployed instances consume disk resources whether they are running or not. Regardless of how the overcloud was shut down, undeleted instances will remain. By default, instances do not automatically restart when the overcloud is stated, but they continue to use the classroom environment's limited disk space. To avoid exercise failures caused by insufficient resources, always delete unneeded instances, as directed in exercise instructions.

- To list all the instances running on the compute nodes, use the following command:

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

- Remove each instance using the listed server instance IDs:

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

- Using the director's undercloud environment, shut down the overcloud using the classroom-only **`rht-overcloud`** script.

```
[student@workstation ~]$ ssh director
(overcloud) [stack@director ~]$ rht-overcloud.sh stop
[*] power is up
[*] node: compute0, state: power on, powering off
[*] node: compute1, state: power on, powering off
[*] node: ceph0, state: power on, powering off
[*] node: computehci0, state: power on, powering off
[*] node: controller0, state: power on, powering off
```

- The overcloud is now shut down. The remaining nodes can now be stopped using procedures specific to your classroom environment.

  View the online course dashboard and each nodes' status text box. Click SHUTDOWN LAB. When all nodes are **STOPPED**, the course environment is cleanly shut down.

Starting the Classroom OpenStack Cluster

The classroom environment is configured to start all VMs automatically, including undercloud, overcloud and support VMs. After the support and undercloud VMs start, the rht-overcloud service starts the overcloud VMs. Since it is possible that VMs might be in an unexpected state after an ungraceful shutdown, manual steps may be needed to manage or restart errant nodes.

The following procedures are required only if problems are detected or some nodes do not start.

- The **`rht-overcloud`** script may be used at any time to start or stop all the overcloud nodes.

```
[student@workstation ~]$ ssh director
(overcloud) [stack@director ~]$ rht-overcloud.sh start
[*] power is up
[*] node: controller0, state: power off, powering on
[*] node: ceph0, state: power off, powering on
[*] node: computehci0, state: power off, powering on
[*] node: compute0, state: power off, powering on
[*] node: compute1, state: power off, powering on
```

- Check the baremetal node power states. Start each node that is not already powered on.

```
(undercloud) [stack@director ~]$ openstack baremetal node list -c Name -c "Power
 State"
+-------------+-------------+
| Name        | Power State |
+-------------+-------------+
| controller0 | power on    |
| compute0    | power on    |
| compute1    | power on    |
| computehci0 | power on    |
| ceph0       | power on    |
+-------------+-------------+
(undercloud) [stack@director ~]$ openstack baremetal node power on  compute1
```

- Check the nodes' server status. If a server is not running, start it. If you determine that a server is running but has an incorrect status, set the correct status as in the example below.

```
(undercloud) [stack@director ~]$ openstack server list -c Name -c Status
+-------------+--------+
| Name        | Status |
+-------------+--------+
| compute1    | ACTIVE |
| compute0    | ACTIVE |
| computehci0 | ACTIVE |
| controller0 | ACTIVE |
| ceph0       | ACTIVE |
+-------------+--------+
(undercloud) [stack@director ~]$ openstack server start compute1
(undercloud) [stack@director ~]$ openstack server set --state active  compute1
```

- If a server has difficulty starting, and does not respond satisfactorily to previous commands to start or set status, the following forced reboot is effective for clearing the current issue, as shown in this example.

```
(undercloud) [stack@director ~]$ openstack server reboot --hard  compute1
```

Repeat these procedures for each problematic node. The overcloud should now be fully started. If you experience further difficulties, fully restting the classroom environment will always result in a working overcloud with all VMs running.

RESETTING THE CLASSROOM ENVIRONMENT

Critical Concept

When the CL210 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 provision a new lab environment.

Resetting 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 following lists the systems never to be reset and those intended to be reset as a group during this course:

Never Need To Be Reset

- `classroom`
- `workstation`
- `power`
- `utility`

Only Reset Together As A Group

- `controller0`
- `compute0`
- `compute1`
- `computehci0`
- `ceph0`
- `director`

Technically, the `director` system is the undercloud. However, in the context of resetting the overcloud, `director` must be included because its services and databases contain control, management, and monitoring information about the overcloud it is managing. Therefore, resetting the overcloud without resetting `director` is to load a fresh overcloud while the `director` retains stale information about the previous overcloud that was just discarded.

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

Click ACTION → Reset for *only* `director` and the undercloud nodes `controller0`, `compute0`, `compute1`, `computehci0`, and `ceph0`.

CAUTION

Never reset the `power` node in the online environment.

If the `power` node is accidentally reset, it will lose the credentials required to power manage the overcloud nodes. The only consistent solution to recover its function is to reprovision the classroom.

If you are still experiencing problems after performing resets, the remaining option is to reprovision the classroom. Resetting *everything* takes time, but results in a fresh environment.

In the online environment, click DELETE LAB, wait, then click PROVISION THE LAB.

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

Students are assigned remote computers in a Red Hat Online Learning classroom. They are accessed through a web application hosted at rol.redhat.com [http://rol.redhat.com]. Students should log in to this site using their Red Hat Customer Portal user credentials.

Controlling the Virtual Machines

The virtual machines in your classroom environment are controlled through a web page. The state of each virtual machine in the classroom is displayed on the page under the Online Lab tab.

Machine States

VIRTUAL MACHINE STATE	DESCRIPTION
STARTING	The virtual machine is in the process of booting.
STARTED	The virtual machine is running and available (or, when booting, soon will be).
STOPPING	The virtual machine is in the process of shutting down.
STOPPED	The virtual machine is completely shut down. Upon starting, the virtual machine boots into the same state as when it was shut down (the disk will have been preserved).
PUBLISHING	The initial creation of the virtual machine is being performed.
WAITING_TO_START	The virtual machine is waiting for other virtual machines to start.

Depending on the state of a machine, a selection of the following actions is available.

Classroom/Machine Actions

BUTTON OR ACTION	DESCRIPTION
PROVISION LAB	Create the ROL classroom. Creates all of the virtual machines needed for the classroom and starts them. Can take several minutes to complete.
DELETE LAB	Delete the ROL classroom. Destroys all virtual machines in the classroom. **Caution: Any work generated on the disks is lost.**
START LAB	Start all virtual machines in the classroom.
SHUTDOWN LAB	Stop all virtual machines in the classroom.
OPEN CONSOLE	Open a new tab in the browser and connect to the console of the virtual machine. Students can log in directly to the virtual machine and run commands. In most cases, students should log in to the `workstation` virtual machine and use **ssh** to connect to the other virtual machines.
ACTION → Start	Start (power on) the virtual machine.
ACTION → Shutdown	Gracefully shut down the virtual machine, preserving the contents of its disk.
ACTION → Power Off	Forcefully shut down the virtual machine, preserving the contents of its disk. This is equivalent to removing the power from a physical machine.
ACTION → Reset	Forcefully shut down the virtual machine and reset the disk to its initial state. **Caution: Any work generated on the disk is lost.**

At the start of an exercise, if instructed to reset a single virtual machine node, click ACTION → Reset for only the specific virtual machine.

At the start of an exercise, if instructed to reset all virtual machines, click ACTION → Reset

If you want to return the classroom environment to its original state at the start of the course, you can click DELETE LAB to remove the entire classroom environment. After the lab has been deleted, you can click PROVISION LAB to provision a new set of classroom systems.

WARNING
The DELETE LAB operation cannot be undone. Any work you have completed in the classroom environment up to that point will be lost.

The Autostop Timer

The Red Hat Online Learning enrollment entitles students to a certain amount of computer time. To help conserve allotted computer time, the ROL classroom has an associated countdown timer, which shuts down the classroom environment when the timer expires.

To adjust the timer, click MODIFY to display the New Autostop Time dialog box. Set the number of hours until the classroom should automatically stop. Click ADJUST TIME to apply this change to the timer settings.

INTERNATIONALIZATION

LANGUAGE SUPPORT

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

PER-USER LANGUAGE SELECTION

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

Language Settings

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

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

NOTE

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

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

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

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

```
[user@host ~]$ LANG=fr_FR.utf8 date
jeu. avril 24 17:55:01 CDT 2014
```

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

Input Method Settings

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

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

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

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

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

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

> **NOTE**
> Any Unicode character can be entered in the GNOME desktop environment if the user knows the character's Unicode code point, by typing **Ctrl**+**Shift**+**U**, followed by the code point. After **Ctrl**+**Shift**+**U** has been typed, an underlined **u** will be displayed to indicate that the system is waiting for Unicode code point entry.
>
> For example, the lowercase Greek letter lambda has the code point U+03BB, and can be entered by typing **Ctrl**+**Shift**+**U**, then **03bb**, then **Enter**.

SYSTEM-WIDE DEFAULT LANGUAGE SETTINGS

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

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

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

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

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

IMPORTANT

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

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

LANGUAGE PACKS

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

REFERENCES

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

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

LANGUAGE CODES REFERENCE

Language Codes

LANGUAGE	$LANG VALUE
English (US)	en_US.utf8
Assamese	as_IN.utf8
Bengali	bn_IN.utf8
Chinese (Simplified)	zh_CN.utf8
Chinese (Traditional)	zh_TW.utf8
French	fr_FR.utf8

LANGUAGE	$LANG VALUE
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

NAVIGATING THE RED HAT OPENSTACK PLATFORM ARCHITECTURE

GOAL

Describe the undercloud and overcloud architecture and service components.

OBJECTIVES

- Describe the classroom environment, support systems, and services.
- Describe the functions of the undercloud components and the management architecture.
- Describe containerized services and management commands.
- Describe the functions of the overcloud components and the management architecture.

SECTIONS

- Introducing the Red Hat OpenStack Platform Infrastructure (and Guided Exercise)
- Describing the Undercloud (and Guided Exercise)
- Introducing Containerized Services (and Guided Exercise)
- Describing the Overcloud (and Guided Exercise)

LAB

- Navigating the Red Hat OpenStack Platform Architecture

INTRODUCING THE RED HAT OPENSTACK PLATFORM INFRASTRUCTURE

OBJECTIVES

After completing this section, students should be able to:

- Describe basic Red Hat OpenStack Platform architecture and terminology.
- Describe the classroom environment and services.

INTRODUCING RED HAT OPENSTACK PLATFORM FUNDAMENTALS

Red Hat OpenStack Platform (RHOSP) uses the term *undercloud* to describe the Director system cloud, which is used to deploy and manage a production cloud. The term *overcloud* describes the production cloud used to deploy and manage end user production workloads.

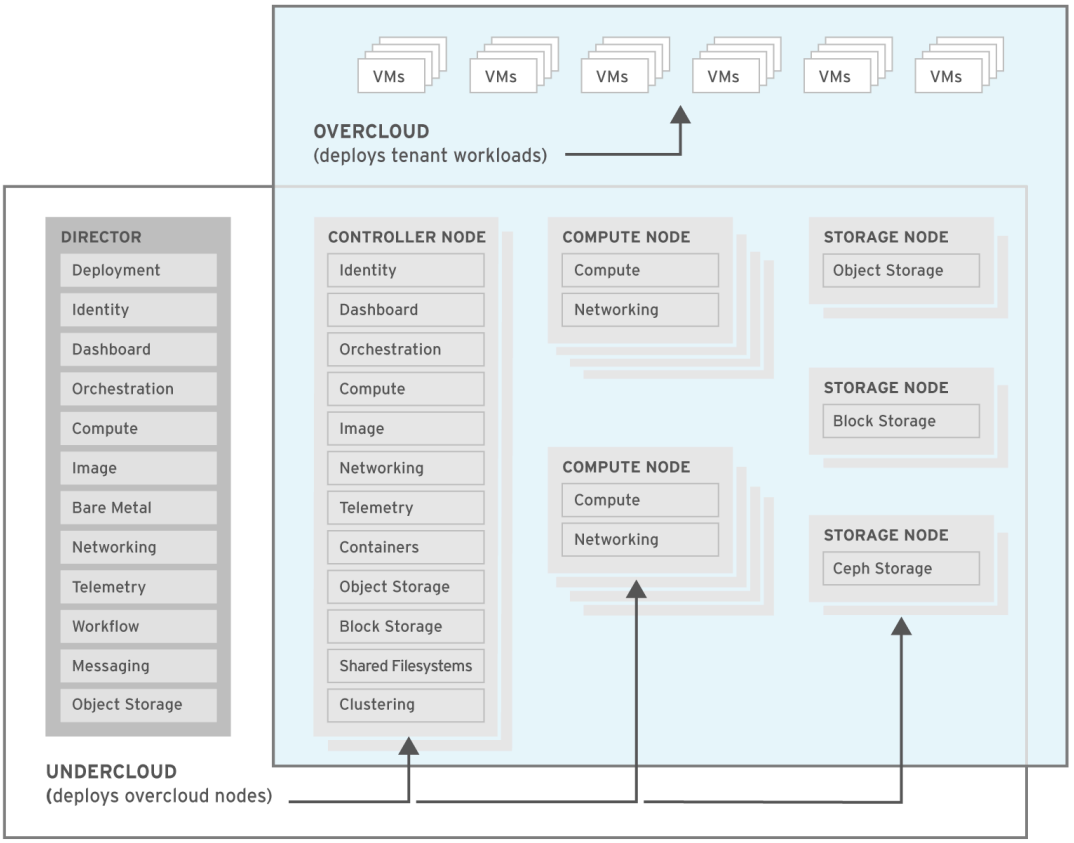


Figure 1.1: An undercloud deploys an overcloud.

The undercloud is the RHOSP Director system itself, plus the provisioning network and resources required to perform undercloud tasks. The overcloud is a RHOSP environment resulting from a template configuration deployed from the undercloud onto available nodes. To access and manage the undercloud, connect to the identity service endpoint of the RHOSP Director system. To access and manage the overcloud, connect to the identity service endpoint on a controller system in the overcloud.

THE CLASSROOM ENVIRONMENT

This classroom has been designed to emulate a production RHOSP environment. Figure 1.2 provides an overview of all external systems, the undercloud, and all overcloud nodes. This narrative section focuses on explaining the external support systems, resources, and services.

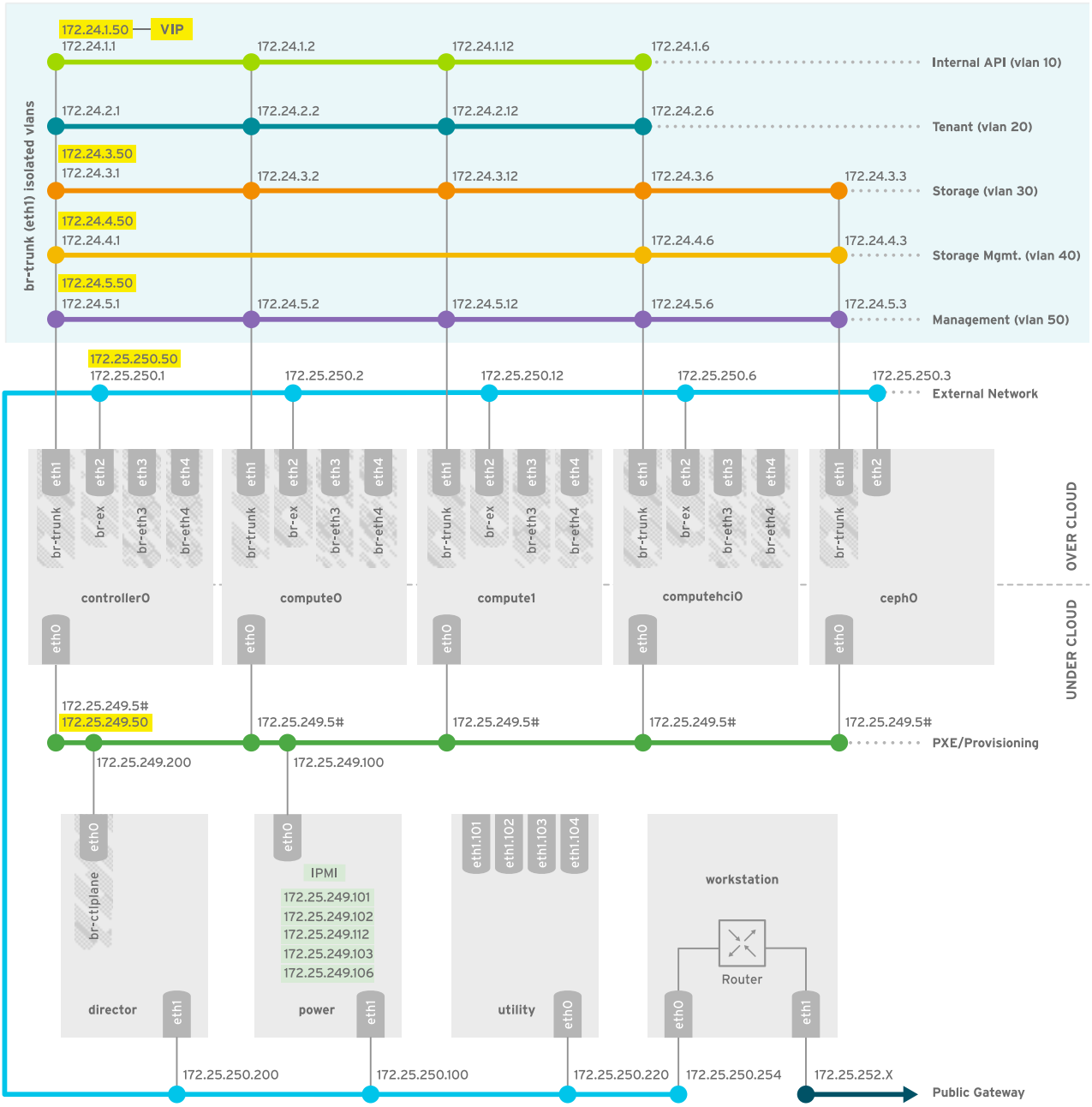


Figure 1.2: Completed classroom overcloud

INTERPRETING THE EXISTING ENVIRONMENT

Private or hybrid Red Hat OpenStack Platform infrastructure is built within an organization's larger enterprise IT environment. Most administrators manage RHOSP installations that were originally designed and built by Red Hat Professional Consulting technicians or that were inherited from previous cloud administrators. During an original RHOSP deployment, the organization's existing network configuration, identity management, IP addressing, and other properties are documented

and used as parameters in the undercloud and overcloud installation tasks. Subsequent RHOSP administrators should familiarize themselves with these relevant configuration elements.

This section introduces the elements of our classroom environment to be understood before managing the classroom RHOSP infrastructure.

Red Hat OpenStack Platform courses are built using a reference architecture design and Red Hat-supported installation methods, meaning that this classroom is a practical example for viewing correct installation design and configuration. However, this course design is constrained by resource limitations. For example, to fit on a single 32 GB workstation-class laptop or desktop, course VMs are allocated a minimally acceptable CPU and RAM size that is less than that recommended for a typical installation. Although typical OpenStack trainings imply that a production installation requires physical compute nodes for robust performance, providing multiple physical systems per student is impractical. Instead, this course's compute nodes are VMs configured with nested virtualization.

In the *Orientation to the Classroom Environment* section, the network design and VM descriptions are provided as an access guide for students. That section included accounts, passwords, system names and addresses, and procedures for cleanly starting and stopping the overcloud. However, while the information is sufficient for self-service users, more information is necessary for understanding the environment from an administrator's viewpoint.

The Organization's Network

The primary classroom network is 172.25.250.0/24. This network is the external, public network from an RHOSP viewpoint. The student's `workstation` VM is on this network, and is configured with a graphical desktop, browser, and RHOSP client utilities, such as the unified command line. As explained in the *Orientation to the Classroom Environment* section, `workstation` is configured as a router between the student's primary network and the 172.25.252.0/24 network, providing access to the `classroom` VM, other student systems, and, by an additional routing through the instructor's system, access to the training provider's network and the internet. If this is an online self-paced course, student environments are self-contained with no instructor system necessary, and no internet access, eliminating that security vulnerability. The following output displays both the student's primary network and the routed classroom network.

```
[student@workstation ~]$ ip a | grep "172.25"
    inet 172.25.250.254/24 brd 172.25.250.255 scope global noprefixroute eth0
    inet 172.25.252.1/24 brd 172.25.252.255 scope global noprefixroute eth1
```

On `director`, interface `eth1` is attached to the external network. The `br-ctlplane` bridge encapsulating interface `eth0` is attached to the `provisioning` network.

```
(undercloud) [stack@director ~]$ ip a | grep "172.25"
    inet 172.25.250.200/24 brd 172.25.250.255 scope global noprefixroute eth1
    inet 172.25.249.200/24 brd 172.25.249.255 scope global br-ctlplane
    inet 172.25.249.202/32 scope global br-ctlplane
    inet 172.25.249.201/32 scope global br-ctlplane
```

RHOSP systems, both undercloud and overcloud nodes, share a number of internal management networks that will be discussed in this course. These include the `tenant` network on which self-service network instances are created, `internal API` communications networks, and the `provisioning` network over which the overcloud is initially deployed. Certain RHOSP administration tools, such as using the Director UI, require access to IP addresses on these RHOSP internal networks. In a production environment, many privileged commands and tools are expected to be run on the Director system, or by use of a dedicated SSH tunnel from a user's desktop system

through Director. Our classroom environment simplifies this need by creating host routes on `workstation`, the only system needing to be granted this elevated level of network access.

```
[student@workstation ~]$ ip route
default via 172.25.252.254 dev eth1 proto static metric 101
172.25.249.50 via 172.25.250.50 dev eth0 proto static metric 100
172.25.249.200 via 172.25.250.200 dev eth0 proto static metric 100
172.25.249.201 via 172.25.250.200 dev eth0 proto static metric 100
172.25.250.0/24 dev eth0 proto kernel scope link src 172.25.250.254 metric 100
172.25.252.0/24 dev eth1 proto kernel scope link src 172.25.252.1 metric 101
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1
```

The `classroom` Server

The VM named `classroom` provides course-specific content when this course is first installed or provisioned for student use. During the course, `classroom` acts as an HTTP file server, providing extra materials for specific exercises. This system also runs the NTP service for all other systems, including the RHOSP nodes, once they are installed. The non-RHOSP systems (`workstation`, `power`, and `utility`), are default Red Hat Enterprise Linux (RHEL) installations which use a `chronyc` client for time management.

```
[student@workstation ~]$ chronyc tracking
Reference ID    : AC19FEFE (classroom.example.com)
Stratum         : 4
Ref time (UTC)  : Tue Sep 18 10:33:56 2018
System time     : 0.000114226 seconds slow of NTP time
...output omitted...
```

The RHOSP systems (for example, the undercloud and overcloud nodes) use the common NTP client; `chronyc` is not installed. Because students do not have login access to the `classroom` system, the time configuration is verified by running NTP commands on the client to verify its server.

```
[heat-admin@controller0 ~]$ sudo ntpq -p
     remote           refid      st t when poll reach   delay   offset  jitter
==============================================================================
*classroom.examp 38.229.71.1      3 u   62  128  377    0.708   -0.063   0.311
```

Prepopulated DNS Zones

The `classroom` system includes a BIND DNS server with name and address entries for systems involved in the initial course preliminary installation tasks. These entries include the base hypervisor system in a physical classroom, the `workstation` systems for all students, and the `classroom` system itself. Although the `classroom` DNS server is not the primary name service for the student environment, it is mentioned here because the student's primary DNS service will be configured to use it for DNS forwarding.

The student's primary DNS service runs on `workstation`. Similar to the use of `workstation` as a router, configuring the student desktop system as the DNS server is not recommended practice, but is a convenience that avoids wasting limited system resources to create an additional VM just to handle routing and name resolution. In a production environment, the organization network gateway and the primary internal DNS server would be on systems other than your personal work desktop. Because only a limited DNS service is required to handle name resolution, without zone transfers or management access control, the service uses `dnsmasq` to loads the contents of **`/etc/hosts`** as the address entries to serve.

A common technique in virtual and containerized environments is to plan the IP addresses and host names to be used by user-launched instances. The majority of the VMs or containers that make up an application have no public-facing elements. They are given arbitrary IP addresses and host names, dynamically assigned when these instances are launched, and known only to other VM and container elements of the application. Because end users do not directly access these instances, no formal host naming is required.

Public-facing applications have a web interface or API where end users connect. These instances use either floating IP addresses on self-service networks or directly assigned IP addresses on provider networks, from an pre-planned allocation pool. Because these addresses are controllable, host names assigned to them are also controllable. In the classroom environment, the address range 172.25.250.101 to .189 is reserved for floating IP addresses. Fully qualified host names for these addresses are assigned in the **`/etc/hosts`** file loaded by `dnsmasq`.

```
[student@workstation ~]$ cat /etc/hosts
127.0.0.1   localhost localhost.localdomain localhost4 localhost4.localdomain4
::1         localhost localhost.localdomain localhost6 localhost6.localdomain6

172.25.254.254 classroom.example.com classroom
172.25.254.254 content.example.com content
172.25.254.254 materials.example.com materials
### rht-vm-hosts file listing the entries to be appended to /etc/hosts

172.25.250.100 power.lab.example.com power
172.25.250.200 director.lab.example.com director
172.25.250.220  utility.lab.example.com utility
# NOTE: The following uses a 249 subnet
172.25.249.201  director-ui.lab.example.com director-ui
172.25.250.254 workstation.lab.example.com workstation

172.25.250.1 controller0.overcloud.example.com controller0
172.25.250.50 keystone.overcloud.example.com keystone
172.25.250.50 dashboard.overcloud.example.com dashboard
172.25.250.2 compute0.overcloud.example.com compute0
172.25.250.3 ceph0.overcloud.example.com ceph0
172.25.250.12 compute1.overcloud.example.com compute1
172.25.250.6 computehci0.overcloud.example.com computehci0

172.25.250.101 float101.instance.example.com float101
172.25.250.102 float102.instance.example.com float102
172.25.250.103 float103.instance.example.com float103
172.25.250.104 float104.instance.example.com float104
...output omitted...
```

The Organization's Identity Management

Many OpenStack courses teach how to create users and groups directly in the OpenStack identity service (keystone) database, allowing administrators to create any projects, users, groups, and roles they deem necessary. In an enterprise environment, however, an organization's users and groups are typically created and managed by a separate security team using an external identity management solution, such as an LDAP server, which can be used as an RHOSP authentication back end. This identity management design is discussed in a later chapter.

In the classroom environment, a Red Hat IdM Server has been configured with all of the users and groups required for this course, similar to a production environment. RHOSP administrators can configure IdM users with project roles so that they can log in to the dashboard and begin work.

However, in this configuration, RHOSP only has read privileges on the IdM back end; it has no modification privileges. If an administrator needs to create a new user, they must contact the IdM administrator or, if the need is unique and temporary, create a new user directly in a different identity service domain in the RHOSP identity service database. The IdM Server runs on the `utility` system. Correct Kerberos credentials are required to query the IdM server.

```
[student@utility ~]$ kinit admin
Password for admin@LAB.EXAMPLE.NET: RedHat123^
[student@utility ~]$ ipa user-find | grep User
  User login: admin
  User login: architect1
  User login: architect2
...output omitted...
  User login: developer1
  User login: developer2
...output omitted...
  User login: operator1
  User login: operator2
...output omitted...
  User login: svc-ldap
```

Similar to user accounts, groups are created and stored in IdM and are available for both dashboard and CLI RHOSP authentication. The following groups have been created for this course:

```
[student@utility ~]$ ipa group-find | grep Group
  Group name: admins
  Group name: consulting-admins
  Group name: consulting-members
  Group name: consulting-swiftoperators
  Group name: editors
  Group name: finance-admins
  Group name: finance-members
  Group name: finance-swiftoperators
  Group name: ipausers
  Group name: production-admins
  Group name: production-members
  Group name: production-swiftoperators
  Group name: research-admins
  Group name: research-members
  Group name: research-swiftoperators
  Group name: trust admins
```

The `utility` server also serves a special network purpose. In a later chapter, you will create provider networks with unique VLAN IDs. In a typical environment, these networks connect to a VLAN-configured enterprise switch. Because this classroom does not include such equipment, the `utility` server is configured to have four virtual NICs representing the VLAN IDs 101 through 104.

```
[student@utility ~]$ ip a | grep eth1
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP
    inet 172.24.250.220/24 brd 172.24.250.255 scope global noprefixroute eth1
4: eth1.101@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 noqueue state UP
    inet 10.0.101.1/24 brd 10.0.101.255 scope global noprefixroute eth1.101
5: eth1.102@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 noqueue state UP
    inet 10.0.102.1/24 brd 10.0.102.255 scope global noprefixroute eth1.102
6: eth1.103@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 noqueue state UP
```

```
    inet 10.0.103.1/24 brd 10.0.103.255 scope global noprefixroute eth1.103
7: eth1.104@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 noqueue state UP
    inet 10.0.104.1/24 brd 10.0.104.255 scope global noprefixroute eth1.104
```

Virtualized Power Management

In a physical data center, organizations typically use enterprise-class equipment that includes a built-in management layer and access interface, supporting out-of-band management protocols such as Intelligent Platform Management Interface (IPMI) and the Scalable Platforms Management API for handling power management. Red Hat courses are constructed using virtual machines that do not have such interfaces, and therefore require emulated power management interactions.

On the `power` system, a CLI interface is installed to create a virtual Baseboard Management Controller (BMC) to manage each RHOSP virtual machine using the IPMI protocol, similar to how real bare-metal machines are managed. Every virtual machine needs its own virtual BMC, each with a different listening address at port 623. IPMI requests directed to the `power` system at the correct IP address initiate a request to the underlying hypervisor for a power state change on the VM represented by that IP address.

```
[root@power ~]$ netstat -uln | grep 623
udp        0      0 172.25.249.103:623      0.0.0.0:*
udp        0      0 172.25.249.101:623      0.0.0.0:*
udp        0      0 172.25.249.102:623      0.0.0.0:*
udp        0      0 172.25.249.112:623      0.0.0.0:*
udp        0      0 172.25.249.106:623      0.0.0.0:*
```

In a physical classroom, students will see the **`virshbmc`** program running. In an online classroom, additional coding was required to operate with the public cloud operator's interface; the program is called **`rhtbmc`**. View the BMC configuration file for the listening IP addresses.

```
[student@power ~]$ cat /etc/bmc/vms
# List the VMs that the fake ipmi will manage. Include in the list one
# VM name per line along with the proxy IP the IPMI client (ipmitool)
# will use to manage it.
#
# EXAMPLE: VM_NAME,IP_ADDRESS
controller0,172.25.249.101
compute0,172.25.249.102
ceph0,172.25.249.103
compute1,172.25.249.112
computehci0,172.25.249.106
```

REFERENCES

Further information is available about *RHOSP Director* at

Red Hat OpenStack Platform Director Life Cycle
https://access.redhat.com/support/policy/updates/openstack/platform/director

TripleO Architecture
https://docs.openstack.org/tripleo-docs/latest/install/introduction/architecture.html

TripleO documentation
https://docs.openstack.org/tripleo-docs/latest/

NAVIGATING THE RED HAT OPENSTACK PLATFORM INFRASTRUCTURE

In this exercise, you will connect to the `workstation`, `utility`, `power`, and `director` nodes to navigate the classroom infrastructure.

OUTCOMES

You should be able to navigate the classroom infrastructure.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab architecture-classroom setup`** command. The script verifies that the required nodes are reachable.

```
[student@workstation ~]$ lab architecture-classroom setup
```

1. On `workstation`, determine which networks and routes exist. The primary classroom network is 172.25.250.0/24. The `workstation` machine is configured as a router between the classroom network and the 172.25.252.0/24 network.

   1.1. List the network interfaces on `workstation`. The `eth0` network interface connects `workstation` to the classroom network, 172.25.250.0/24. The `eth1` network interface connects to the 172.25.252.0/24 network.

```
[student@workstation ~]$ ip a
...output omitted...
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether 52:54:00:00:fa:fe brd ff:ff:ff:ff:ff:ff
    inet 172.25.250.254/24 brd 172.25.250.255 scope global noprefixroute eth0
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe00:fafe/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether 52:54:00:01:fc:00 brd ff:ff:ff:ff:ff:ff
    inet 172.25.252.250/24 brd 172.25.252.255 scope global noprefixroute eth1
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe01:fc00/64 scope link
       valid_lft forever preferred_lft forever
...output omitted...
```

   1.2. View the routing rules on `workstation`.

   The `eth0` network device connects to the RHOSP cloud deployment. The `eth1` network device provides routing to access the `classroom` VM and other systems.

```
[student@workstation ~]$ ip route
default via 172.25.252.254 dev eth1 proto static metric 103
172.25.249.50 via 172.25.250.50 dev eth0 proto static metric 102
172.25.249.200 via 172.25.250.200 dev eth0 proto static metric 102
172.25.249.201 via 172.25.250.200 dev eth0 proto static metric 102
172.25.250.0/24 dev eth0 proto kernel scope link src 172.25.250.254 metric 102
172.25.252.0/24 dev eth1 proto kernel scope link src 172.25.252.250 metric 103
192.168.122.0/24 dev virbr0 proto kernel scope link src 192.168.122.1
```

2. To properly synchronize services among RHOSP nodes, the controller node must reference more accurate NTP servers. Verify the NTP service provided by the `classroom` VM.

```
[student@workstation ~]$ chronyc sources
210 Number of sources = 1
MS Name/IP address         Stratum Poll Reach LastRx Last sample
===============================================================================
^* classroom.example.com       8  10   377   787    -86us[ -101us] +/-  139us
```

3. To selectively assign host names to instances connected to an external provider network, DNS services are must be configured for use by the RHOSP deployment. The fully qualified host names are assigned in the **`/etc/hosts`** file. The file contents are loaded by the `dnsmasq` service.

3.1. Verify the DNS service provided by the `dnsmasq` service on `workstation`.

```
[student@workstation ~]$ ps -ef | grep dnsmasq
...output omitted...
root 1614  1613  0 09:27 ? 00:00:00 /usr/sbin/dnsmasq --conf-file=/var/
lib/libvirt/dnsmasq/default.conf --leasefile-ro --dhcp-script=/usr/libexec/
libvirt_leaseshelper
```

3.2. View the **`/etc/hosts`** file that translates host names to IP addresses and is loaded by the `dnsmasq` service. The file contains the entries of all the virtual machines used in the classroom and also instances launched on the overcloud.

```
[student@workstation ~]$ cat /etc/hosts
...output omitted...
172.25.250.100 power.lab.example.com power
172.25.250.200 director.lab.example.com director
172.25.250.220 utility.lab.example.com utility
# NOTE: The following uses a 249 subnet
172.25.249.201 director-ui.lab.example.com director-ui
172.25.250.254 workstation.lab.example.com workstation

172.25.250.1 controller0.overcloud.example.com controller0
172.25.250.50 keystone.overcloud.example.com keystone
172.25.250.50 dashboard.overcloud.example.com dashboard
172.25.250.2 compute0.overcloud.example.com compute0
172.25.250.3 ceph0.overcloud.example.com ceph0
172.25.250.12 compute1.overcloud.example.com compute1
172.25.250.6  computehci0.overcloud.example.com computehci0

172.25.250.101 float101.instance.example.com float101
```

```
172.25.250.102 float102.instance.example.com float102
172.25.250.103 float103.instance.example.com float103
172.25.250.104 float104.instance.example.com float104
...output omitted...
```

4. On the `utility` node, verify the organization's users and groups that exists on a Red Hat IdM Server. RHOSP administrators can configure IdM users with project roles so they can log in to the dashboard and begin work.

4.1. Use the **`ssh`** command to log in to `utility` as the `student` user. Use the **`kinit admin`** command to get the Kerberos tokens.

```
[student@workstation ~]$ ssh utility
[student@utility ~]$ kinit admin
Password for admin@LAB.EXAMPLE.NET: RedHat123^
```

4.2. Use the **`ipa user-find`** command to list the user accounts. The `svc-ldap` account is the integration user. The RHOSP identity service uses the `svc-ldap` account to query other IdM server user accounts.

```
[student@utility ~]$ ipa user-find | grep User
  User login: admin
  User login: architect1
  User login: architect2
  User login: architect3
  ...output omitted...
  User login: developer1
  User login: developer2
  User login: developer3
  ...output omitted...
  User login: operator1
  User login: operator2
  User login: operator3
  ...output omitted...
  User login: svc-ldap
```

4.3. Use the **`ipa group-find`** command to list the group accounts.

```
[student@utility ~]$ ipa group-find | grep Group
  Group name: admins
  Group name: consulting-admins
  Group name: consulting-members
  Group name: consulting-swiftoperators
  Group name: editors
  Group name: finance-admins
  Group name: finance-members
  Group name: finance-swiftoperators
  Group name: ipausers
  Group name: production-admins
  Group name: production-members
  Group name: production-swiftoperators
  Group name: research-admins
  Group name: research-members
  Group name: research-swiftoperators
```

```
  Group name: trust admins
```

Log off from `utility`.

5. On the `power` node, verify that the **`rhtbmc`** program is running. View the virtual BMC configuration file. View the IPMI request listening addresses found to be using port 623.

5.1. Use the **`ssh`** command to log in to `power` then become the `root` user.

```
[student@workstation ~]$ ssh power
[student@power ~]$ sudo -i
[root@power ~]#
```

5.2. Verify that the **`rhtbmc`** program is running.

```
[root@power ~]# ps -ef | grep rhtbmc
... /usr/bin/python /usr/bin/rhtbmc
... /usr/bin/python /usr/bin/rhtbmc
... /usr/bin/python /usr/bin/rhtbmc
... /usr/bin/python /usr/bin/rhtbmc
... /usr/bin/python /usr/bin/rhtbmc
... /usr/bin/python /usr/bin/rhtbmc
... grep --color=auto rhtbmc
```

5.3. View the virtual BMC configuration file, **`/etc/bmc/vms`**. The configuration file lists the virtual machines and their corresponding IP addresses that are used to listen for IPMI requests.

```
[root@power ~]# cat /etc/bmc/vms
# List the VMs that the fake ipmi will manage. Include in the list one
# VM name per line along with the proxy IP the IPMI client (ipmitool)
# will use to manage it.
#
# EXAMPLE: VM_NAME,IP_ADDRESS
controller0,172.25.249.101
compute0,172.25.249.102
ceph0,172.25.249.103
computehci0,172.25.249.106
compute1,172.25.249.112
```

5.4. List the IP addresses that are used by the virtual BMC to listen for IPMI requests on port 623.

```
[root@power ~]# netstat -uln | grep 623
udp        0      0 172.25.249.112:623      0.0.0.0:*
udp        0      0 172.25.249.106:623      0.0.0.0:*
udp        0      0 172.25.249.103:623      0.0.0.0:*
udp        0      0 172.25.249.102:623      0.0.0.0:*
udp        0      0 172.25.249.101:623      0.0.0.0:*
```

Log off from `power`.

6. Test the IPMI emulation software, which is performing power management for the overcloud's virtual machine nodes. On `director`, use the **`ipmitool`** command to query the status of the `compute1` node. The IPMI address for `compute1` is 172.25.249.112.

   6.1. Use the **`ssh`** command to log in to `director` as the `stack` user.

```
[student@workstation ~]$ ssh stack@director
(undercloud) [stack@director ~]$
```

   6.2. Use the IPMI command-line tool to check the current power status of `compute1`.

```
(undercloud) [stack@director ~]$ ipmitool -I lanplus -U admin -P password \
-H 172.25.249.112 power status
Chassis Power is on
```

   Log off from `director`.

This concludes the guided exercise.

DESCRIBING THE UNDERCLOUD

OBJECTIVES

After completing this section, students should be able to:

- Describe the core Red Hat OpenStack components performing undercloud services.
- Manage undercloud services.
- View the undercloud structure.

INTRODUCING THE UNDERCLOUD

Red Hat OpenStack Platform (RHOSP) Director is a standalone OpenStack all-in-one installation, providing a tool set for installing and managing a complete OpenStack infrastructure environment. It is based primarily on the OpenStack Deployment component developed in the TripleO project, which is an abbreviation for "OpenStack on OpenStack". The undercloud uses OpenStack core components running on the dedicated all-in-one installation (the *undercloud*) to install an operational OpenStack cloud (the overcloud). It uses these core components, plus new components, to locate, provision, deploy, and configure bare-metal systems as OpenStack controller, compute, networking and storage nodes.

Red Hat OpenStack Platform Director is a deployment cloud for OpenStack infrastructure, in which the cloud workload is the overcloud systems themselves: controllers, compute nodes, and storage nodes. Because infrastructure nodes are commonly built directly on physical hardware systems, the undercloud may be referred to as a *bare-metal cloud*. However, as you will experience in this course, an undercloud can deploy infrastructure to virtual systems for learning, testing, and specific use cases. Similarly, overclouds almost exclusively deploy virtual machines and containers but can be used to deploy tenant workloads directly to dedicated, physical systems, such as blade servers or enterprise rack systems, by incorporating bare-metal drivers and methods. Therefore, the terms *bare-metal cloud* and *tenant workload cloud* are only a convenient frame of reference.

The components that form the undercloud provide multiple functions:

Identity Service (keystone)
: The Identity service provides user authentication and authorization to the OpenStack components running on the undercloud.

Image Service (glance)
: The Image service stores the software to be installed on the bare-metal server. The image must contain all the required software to boot an operating system and the run the KVM hypervisor and the compute service.

Compute Service (nova)
: The Compute service integrates with the bare-metal service to provision bare-metal machines in the same way as virtual machines, by first introspecting the physical machines to obtain hardware attributes and configuration. The compute service also provides scheduling facilities. For each request to deploy a new bare-metal node with a specific overcloud role, the compute service filters the list of available bare-metal nodes, ensuring that the selected node meets the hardware requirements.

Bare Metal Service (ironic)
: The bare-metal service provisions the bare-metal nodes. The service retrieves the image provided by the image service associated with a specific overcloud role and boots the bare-

metal node with the deployment ramdisk. To customize the images, the bare-metal service also provides the **diskimage-builder** tool. An associated service named `ironic-inspector` is used for introspecting the hardware of unregistered bare-metal nodes. The undercloud uses an out-of-band management interface, usually Intelligent Platform Management Interface (IPMI), to manage, monitor, and configure bare-metal systems.

Orchestration Service (heat)
The Orchestration service provides a set of YAML templates to define configuration and provisioning instructions to deploy OpenStack infrastructure servers. The orchestration service defines overcloud roles to provision overcloud nodes. Templates act as a set of blueprints for the overcloud deployment. 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.

Object store (swift)
The object store provides temporary storage to the undercloud for storing user images, deployment logs, and inspection data.

OpenStack Networking Service (neutron)
The networking service configures network interfaces that are required on the undercloud. The undercloud requires at lest two networks: `provisioning` and `external`. The `provisioning` network provides DHCP and PXE boot functions to register bare-metal servers to use as overcloud nodes. Th `external` network provide remote connectivity to all nodes.

DISCUSSING UNDERCLOUD SERVICES

The undercloud consists of 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. When the RHOSP Director is performing the build process for the overcloud, the nodes being provisioned to be controller, compute, network, and storage systems are considered as the workload of the undercloud. When deployment and configuration are complete, these nodes reboot to become the overcloud.

Viewing the Identity Service on the Undercloud

When the undercloud installation completes it creates the **`stackrc`** identity environment file in the user's home directory with the `admin` credentials. The passwords for all the user accounts required for the undercloud are stored in the **`undercloud-passwords.conf`** file. You must source the **`stackrc`** file to interact with the undercloud services using the OpenStack CLI client. Red Hat OpenStack Platform Director installation deploys the undercloud and adds the **`source stackrc`** command in the **`.bashrc`** file. In the output below, the **`stackrc`** file is sourced to execute the **`openstack service list`** command on the undercloud.

```
(undercloud) [stack@director ~]$ openstack service list
+----------------------------------+------------------+-------------------------+
| ID                               | Name             | Type                    |
+----------------------------------+------------------+-------------------------+
| 075dd44c05974b48aff6f6fa53424dd7 | mistral          | workflowv2              |
| 0de77e931a854bf6a6238ce7d42a3d64 | neutron          | network                 |
| 1a834ff0bb514a269ff556ca67947b8a | heat-cfn         | cloudformation          |
| 2bec65fe01074fe6aa41af623c615e1f | swift            | object-store            |
| 5185fedb14c140f693f47e60f9bba2b7 | zaqar-websocket  | messaging-websocket     |
| 60ba8527e4e74a058a3858fb9774bf5d | placement        | placement               |
| 6d44bbf3d4974f91be0d4ec9f30e94f0 | ironic           | baremetal               |
| 7190b08b2af64b7c9d528619fa37ed23 | ironic-inspector | baremetal-introspection |
| 9a17f943d0ff4ee8b8774e6e6febe455 | heat             | orchestration           |
| a4f255e60a204de09e1ccbd347d68d40 | zaqar            | messaging               |
```

```
| baa06a7344bb4e17b0239ad9a79a0dc5 | nova             | compute                 |
| f5cf1cbe40a9426ead51134bb8aeabcc | keystone         | identity                |
| ff23d031c3bf41f3a79785feabf0dd19 | glance           | image                   |
+----------------------------------+------------------+-------------------------+
```

In the classroom environment, the OS_AUTH_URL environment variable in the **stackrc** file is set to the public endpoint of the identity service running on the undercloud. The classroom environment uses SSL for all public endpoints for the OpenStack services on the undercloud.

```
(undercloud) [stack@director ~]$ openstack endpoint list \
-c 'Service Type' -c 'Interface' -c 'URL'
+-------------------------+-----------+-----------------------------+
| Service Type            | Interface | URL                         |
+-------------------------+-----------+-----------------------------+
...output omitted...
| identity                | public    | https://172.25.249.201:13000 |
+-------------------------+-----------+-----------------------------+
(undercloud) [stack@director ~]$ grep "^OS_AUTH_URL" stackrc
OS_AUTH_URL=https://172.25.249.201:13000/
```

Identifying the Networking Service on the Undercloud

To provision overcloud nodes, the undercloud is configured with a provisioning network that provides DHCP and PXE boot functions to set up the bare-metal nodes. The provisioning network is a large-capacity, dedicated, and isolated network, separate from the normal public network. After an overcloud deployment, Red Hat OpenStack Platform Director continues to manage and update the overcloud across this isolated, secure provisioning network, completely segregated from both external and internal OpenStack traffic.

Viewing the Provisioning Network

View the **undercloud.conf** file, created to build the undercloud, to verify the provisioning network. In the output below, the DHCP address range from **dhcp_start** to **dhcp_end** is the scope for the OpenStack networking dnsmasq service managing the provisioning subnet. Nodes deployed to the provisioning network are assigned an IP address from this scope for their provisioning NIC. The **inspection_iprange** is the scope for the bare-metal dnsmasq service, used to temporarily assign addresses to registered nodes during PXE boot at the start of the introspection process.

```
(undercloud) [stack@director ~]$ cat undercloud.conf | egrep -v "(^#.*|^$)"
[DEFAULT]
local_ip = 172.25.249.200/24
undercloud_public_vip = 172.25.249.201
undercloud_admin_vip = 172.25.249.202
undercloud_ntp_servers = 172.25.254.254
dhcp_start = 172.25.249.51
dhcp_end = 172.25.249.59
inspection_iprange = 172.25.249.150,172.25.249.180
masquerade_network = 172.25.249.0/24
undercloud_service_certificate = /etc/pki/tls/certs/undercloud.pem
generate_service_certificate = false
local_interface = eth0
network_cidr = 172.25.249.0/24
network_gateway = 172.25.249.200
hieradata_override = /home/stack/hieradata.txt
undercloud_debug = false
```

```
enable_telemetry = false
enable_ui = true
[auth]
undercloud_admin_password = redhat
[ctlplane-subnet]
```

View the undercloud's configured network interfaces. The `br-ctlplane` bridge is the `172.25.249.0` provisioning network. The `eth1` interface is the `172.25.250.0` public network.

```
(undercloud) [stack@director ~]$ ip a | grep -r 'br-ctlplane|eth1'
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    inet 172.25.250.200/24 brd 172.25.250.255 scope global noprefixroute eth1
6: br-ctlplane: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UNKNOWN group default qlen 1000
    inet 172.25.249.200/24 brd 172.25.249.255 scope global br-ctlplane
    inet 172.25.249.202/32 scope global br-ctlplane
    inet 172.25.249.201/32 scope global br-ctlplane
```

The provisioning subnet is configured for DHCP. The networking service configures a `dnsmasq` instance to manage the subnet scope. The configured DNS nameserver value is handed out as a DHCP option when instances deployed on this subnet query `dnsmasq` to obtain an IP address and a default gateway.

```
(undercloud) [stack@director ~]$ openstack subnet show ctlplane-subnet
+-------------------+-----------------------------+
| Field             | Value                       |
+-------------------+-----------------------------+
| allocation_pools  | 172.25.249.51-172.25.249.59 |
| cidr              | 172.25.249.0/24             |
| created_at        | 2018-09-26T00:38:29Z        |
| description       |                             |
| dns_nameservers   | 172.25.250.254              |
| enable_dhcp       | True                        |
| gateway_ip        | 172.25.249.200
...output omitted...
```

Accessing the Image Service on the Undercloud

When bare-metal nodes are first provisioned, the `bare metal` service use IPMI commands to power on the nodes. The nodes are configured to PXE boot by default. During the PXE boot, each bare-metal node requests a DHCP address, a kernel image, and a ramdisk image to perform an initial network boot. These kernel and ramdisk boot images are requested from the undercloud image service. The following output lists the images required for PXE-booting the bare-metal node for hardware introspection:

```
(undercloud) [stack@director ~]$ openstack image list
+--------------------------------------+------------------------+--------+
| ID                                   | Name                   | Status |
+--------------------------------------+------------------------+--------+
| 525385d3-7f95-4911-be6e-ea205db379a8 | bm-deploy-kernel       | active |
| 246a9a84-f097-4ce1-82d0-e8a754ef94b2 | bm-deploy-ramdisk      | active |
...output omitted...
```

The bare-metal node setup used during the overcloud deployment uses the `overcloud-full-initrd` and the `overcloud-full-vmlinuz` images to perform a network boot. The `overcloud-full` image is then copied to the overcloud node's boot disk. The `overcloud-full` image is a working Red Hat Enterprise Linux system with all of the Red Hat OpenStack Platform and Red Hat Ceph Storage packages already installed but not configured.

```
(undercloud) [stack@director ~]$ openstack image list
+--------------------------------------+------------------------+--------+
| ID                                   | Name                   | Status |
+--------------------------------------+------------------------+--------+
...output omitted...
| 67559cdd-b5ac-4a65-8660-e1d6b1b3d8bb | overcloud-full         | active |
| a9a01686-8712-4a2b-9d00-a97984cfd1f6 | overcloud-full-initrd  | active |
| 9b42fff1-74e8-476f-b28a-cbc4b344fcda | overcloud-full-vmlinuz | active |
+--------------------------------------+------------------------+--------+
```

Identifying the Bare Metal Service on the Undercloud

The bare metal service registers the nodes to be used to deploy the overcloud. The workflow service (mistral) manages this registration task set, allowing multiple tasks and actions to occur simultaneously. When preparing to build a new overcloud, hardware introspection is performed using the `bm-deploy-ramdisk`, and `bm-deploy-kernel` images. These two images work together to boot the prospective bare-metal nodes to gather and report the node capabilities back to the undercloud's bare metal service, which updates its database and stores the node information using the object store service.

All the registered bare-metal servers are available for the overcloud deployment. Use the **`openstack baremetal node list`** command to list the registered bare-metal servers.

```
(undercloud) [stack@director ~]$ openstack baremetal node list -c Name -c 'Power
 State' -c 'Provisioning State'
+-------------+-------------+--------------------+
| Name        | Power State | Provisioning State |
+-------------+-------------+--------------------+
| controller0 | power on    | active             |
| compute0    | power on    | active             |
| computehci0 | power on    | active             |
| compute1    | power on    | active             |
| ceph0       | power on    | active             |
+-------------+-------------+--------------------+
```

Viewing the Orchestration Service on the Undercloud

Red Hat OpenStack Platform Director ships with a full set of working overcloud templates, including many optional configuration environment files, in the **`/usr/share/openstack-tripleo-heat-templates/`** directory. The orchestration service manages the deployment process by communication the resource-building instructions and Puppet scripts that configure the overcloud node based on its assigned deployment role. Overcloud deployment roles are assigned in the orchestration templates by including scheduler hints that instruct the orchestration engine how to select nodes for each assigned role.

Use the **`openstack stack list`** command to query the undercloud about the successful deployment of the overcloud orchestration stack.

```
(undercloud) [stack@director ~]$ openstack stack list -c 'Stack Name' -c 'Stack
 Status'
```

```
+------------+-----------------+
| Stack Name | Stack Status    |
+------------+-----------------+
| overcloud  | CREATE_COMPLETE |
+------------+-----------------+
```

Viewing Compute Services on the Undercloud

In order for the stack to be deployed, the orchestration service makes successive calls to the compute service running on the undercloud. The compute service depends upon the bare metal service, which, as described earlier, creates an inventory of introspected hardware. For each request to deploy a new overcloud node with a specific role, the compute scheduler filters the list of available nodes, ensuring that the selected nodes meet the hardware requirements. The following output shows a list of the compute services running on the undercloud:

```
(undercloud) [stack@director ~]$ openstack compute service list
+----+----------------+--------------------------+----------+---------+-------+
| ID | Binary         | Host                     | Zone     | Status  | State |
+----+----------------+--------------------------+----------+---------+-------+
|  1 | nova-conductor | director.lab.example.com | internal | enabled | up    |
|  2 | nova-scheduler | director.lab.example.com | internal | enabled | up    |
|  5 | nova-compute   | director.lab.example.com | nova     | enabled | up    |
+----+----------------+--------------------------+----------+---------+-------+
```

POWER MANAGEMENT ON THE UNDERCLOUD

The nodes to be deployed are typically bare-metal physical systems, such as blade servers or rack systems, with physical network management interfaces for remote power off access and administration. Access each node to verify that the systems are configured with multiple NICs, and the correct configuration of CPU, RAM, and hard disk space for the assigned deployment role. In this course, the nodes are virtual machines with a small-scale configuration.

Power management in a cloud environment normally uses the IPMI management NIC built into a server chassis. However, virtual machines do not normally have a lights-out-management platform interface. Instead, they are controlled by the appropriate virtualization management software, which connects to the running virtual machine's hypervisor to request power management actions and events. In this classroom, a Baseboard Management Controller (BMC) emulator runs on the `power` virtual machine, configured with one unique IP address per virtual machine node. Upon receiving a valid IPMI request at the correct listener, the BMC emulator sends the request to the hypervisor, which performs the request on the corresponding virtual machine. The **`instackenv-initial.json`** file defines the MAC address, IPMI address, power management user name and password, for each bare-metal node to be registered. This node registration file can be in either JSON or YAML format. The following example shows the **`instackenv-initial.json`** configuration file in JSON format.

```
(undercloud) [stack@director ~]$ cat instackenv-initial.json
{
  "nodes": [
    {
      "name": "controller0",
      "arch": "x86_64",
      "cpu": "2",
      "disk": "40",
      "memory": "8192",
      "mac": [ "52:54:00:00:f9:01" ],
      "pm_addr": "172.25.249.101",
```

```
        "pm_type": "pxe_ipmitool",
        "pm_user": "admin",
        "pm_password": "password",
        "pm_port": "623",
        "capabilities": "node:controller0,boot_option:local"
      },
...output omitted...
```

Performing IPMI Power Management

The `power` virtual machine acts like an IPMI hardware layer for each of the bare-metal nodes. Properly structured IPMI commands sent to a listener are translated by the IPMI emulator into requests for the underlying hypervisor system, which performs the action on the requested node.

IPMI IP Addresses in the Classroom

MACHINE NAME	IP ADDRESS ON THE PROVISIONING NETWORK	VIRTUAL IP ADDRESS ON THE POWER IPMI EMULATOR
`controller0`	172.25.249.1	172.25.249.101
`compute0`	172.25.249.2	172.25.249.102
`compute1`	172.25.249.12	172.25.249.112
`computehci0`	172.25.249.6	172.25.249.106
`ceph0`	172.25.249.3	172.25.249.103

The classroom does not require the full IPMI set of capabilities, only the ability to power cycle or start nodes programmatically on demand. The command-line utility to test the functionality of the power IPMI emulation uses this syntax:

```
(undercloud) [stack@director ~]$ ipmitool -I lanplus \
-U admin -P password -H IP power status|on|off
```

The **`-I`** interface options are compiled into the command and may be seen with **`ipmitool -h`**. The **`lanplus`** option indicates the use of the **`IPMI v2.0 RMCP+ LAN`** interface.

For example, run the following command to view the power status of the `controller0` node.

```
(undercloud) [stack@director ~]$ ipmitool -I lanplus \
-U admin -P password -H 172.25.249.101 power status
Chassis Power is on
```

The OpenStack CLI interacts with the bare metal service using the **`openstack baremetal`** command for operations on bare-metal nodes registered to the bare metal service. Use the **`openstack baremetal node power on`** command to power on a registered bare-metal node.

```
(undercloud) [stack@director ~]$ openstack baremetal node power on controller0
```

Similarly use the **`openstack baremetal node power off`** command to power off a registered bare-metal node.

```
(undercloud) [stack@director ~]$ openstack baremetal node power off controller0
```

IMPORTANT

In the default bare metal service configuration the power state of the bare-metal nodes are not synchronized with the power state of overcloud nodes. This is because in a high availability (HA) setup, Pacemaker is responsible for controlling the power state of the overcloud nodes for fencing purposes. In a non-HA configuration, set the **`force_power_state_during_sync`** parameter to **`True`** in the **`/etc/ironic/ironic.conf`** file to force the bare metal service to control the power state of the bare-metal nodes. On the undercloud, set the **`sync_power_state_interval`** parameter in the **`/etc/nova/nova.conf`** file to a value, in seconds, for how often to check and sync the power state. The default undercloud installation sets the **`force_power_state_during_sync`** parameter to **`False`**.

UNDERCLOUD RECOMMENDED PRACTICES

Red Hat OpenStack Platform Director is a tool used to install and manage the deployment and life cycle of Red Hat OpenStack Platform 7 (Kilo) and later versions. It is targeted for cloud operator use cases where managed updates, upgrades, and infrastructure control are critical for underlying OpenStack operations. It also provides an API-driven framework providing hardware introspection, environment monitoring, capacity planning, utilization metrics, service allocation, and stack management.

Life-cycle management for cloud infrastructure has operational tasks similar to legacy enterprise management, but also incorporates new interpretations of Continuous Integration and DevOps. The cloud industry differentiates stages of life-cycle management by categorizing tasks as *Day 0 (Planning)*, *Day 1 (Deploying)*, and *Day 2 (Operations)*.

- *Planning*: introspection, network topology, service parameters, resource capacity.
- *Deployment*: deployment orchestration, service configuration, sanity checks, testing.
- *Operations*: updates and upgrades, scaling up and down, change management, compliance.

As a Day 0 Planning tool, Director provides default, customizable configuration files to define cloud architecture, including networking and storage topologies, OpenStack service parameters, and third-party plug-in integration. These default files and templates implement Red Hat's highly available reference architecture and recommended practices.

Director is most commonly recognized as a Day 1 deployment tool, performing orchestration, configuration, and validation for building overclouds. Tasks include hardware preparation, software deployment using Puppet manifests, and Tempest validation scripts, making it easier on operators to learn and implement customizations within the Director framework as a recommended practice for consistency and reusability.

Director is designed as a central management system for ongoing Day 2 Operations. It can perform environment health checks, auto-scale an overcloud by adding or replacing nodes, perform minor release updates and major version upgrades, plus patching, monitoring, and regulation compliance.

To use the undercloud for Day 2 management, all management must be performed using the undercloud CLI or APIs. Currently, there is no reasonable expectation that the undercloud can detect, interpret, or reconcile manual changes not implemented through the undercloud. Using outside tool sets loses the ability to perform safe and predictable updates, upgrades, and scaling.

Integration with third-party tools that exclusively call undercloud APIs is recommended, and does not break Day 2 operation support. Recommended examples include integration between the undercloud and Red Hat CloudForms, Red Hat Satellite Server, and Red Hat Ansible Tower.

The undercloud includes tools that help with hardware testing, and is architected to facilitate future functionality for automated OpenStack upgrades and patch management, centralized log collection, and diagnostics.

REFERENCES

Further information about the Director installation is available at *Director Installation & Usage guide for Red Hat OpenStack Platform 13*
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/director_installation_and_usage/

Further information about IPMI is available at *Intelligent Platform Management Interface Specification*
https://www.intel.com/content/www/us/en/servers/ipmi/ipmi-second-gen-interface-spec-v2-rev1-1.html

VIEWING THE UNDERCLOUD ARCHITECTURE

In this exercise, you will connect to the undercloud node, `director`, to navigate the undercloud configuration.

OUTCOMES

You should be able to navigate the undercloud.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab architecture-undercloud setup`** command. The script verifies that the undercloud node, `director`, is reachable.

```
[student@workstation ~]$ lab architecture-undercloud setup
```

1. Log in to the `director` node as the `stack` user. Open the **`undercloud.conf`** file and inspect the DHCP configuration.

   1.1. From `workstation`, log in to the `director` node.

```
[student@workstation ~]$ ssh stack@director
(undercloud) [stack@director ~]$
```

   1.2. Open the **`/home/stack/undercloud.conf`** file. Notice the IP addresses in the **`dhcp_start`** and **`dhcp_end`** parameters.

```
(undercloud) [stack@director ~]$ grep "^dhcp*" undercloud.conf
dhcp_start = 172.25.249.51
dhcp_end = 172.25.249.59
```

   1.3. In the **`/home/stack/undercloud.conf`** file, search for the **`undercloud_public_vip`** and **`undercloud_admin_vip`** parameters. These IP addresses are defined for the `director`'s public API and admin API endpoints.

```
(undercloud) [stack@director ~]$ grep "^undercloud_.*vip" undercloud.conf
undercloud_public_vip = 172.25.249.201
undercloud_admin_vip = 172.25.249.202
```

   1.4. In the **`/home/stack/undercloud.conf`** file, search for the **`undercloud_admin_password`** parameter.

```
(undercloud) [stack@director ~]$ grep "^undercloud_.*password" undercloud.conf
undercloud_admin_password = redhat
```

2. Observe that the **stackrc** environment file automatically loaded with the admin user credentials. View the stack user's environment, which is used to connect to the undercloud.

   2.1. Use the **openstack endpoint list** command to list the public API endpoint of the identity service on director.

```
(undercloud) [stack@director ~]$ openstack endpoint list \
-c 'Service Type' -c 'Interface' -c 'URL'
+-------------------------+-----------+-----------------------------+
| Service Type            | Interface | URL                         |
+-------------------------+-----------+-----------------------------+
...output omitted...
| identity                | public    | https://172.25.249.201:13000 |
+-------------------------+-----------+-----------------------------+
```

   2.2. View the identity service environment variables loaded in the stack user's environment.

```
(undercloud) [stack@director ~]$ env | grep OS_
OS_BAREMETAL_API_VERSION=1.34
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_NAME=admin
OS_IDENTITY_API_VERSION=3
OS_PASSWORD=redhat
OS_AUTH_TYPE=password
PS1=${OS_CLOUDNAME:+($OS_CLOUDNAME)} [\u@\h \W]\$
OS_AUTH_URL=https://172.25.249.201:13000/
OS_USERNAME=admin
OS_NO_CACHE=True
OS_CLOUDNAME=undercloud
OS_PROJECT_DOMAIN_NAME=Default
```

   Notice that the OS_AUTH_URL environment variable is set to the public identity API endpoint. The **stackrc** file loads the admin user's credentials. The OS_PASSWORD variable is set to **redhat** as specified in the **undercloud.conf** file. The OS_CLOUDNAME variable is set to **undercloud**.

3. View the provisioning and external network interfaces configured on the director node. List the details for the ctlplane-subnet subnet.

   3.1. List the provisioning and external networks on the director node. The br-ctlplane bridge is the 172.25.249.0 provisioning network; the eth1 interface is the 172.25.250.0 public network.

```
(undercloud) [stack@director ~]$ ip a | grep -E 'br-ctlplane|eth1'
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
group default qlen 1000
inet 172.25.250.200/24 brd 172.25.250.255 scope global noprefixroute eth1
6: br-ctlplane: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
UNKNOWN group default qlen 1000
inet 172.25.249.200/24 brd 172.25.249.255 scope global br-ctlplane
inet 172.25.249.202/32 scope global br-ctlplane
```

```
inet 172.25.249.201/32 scope global br-ctlplane
```

3.2. Use the **`openstack subnet list`** command to list the undercloud subnet on the provisioning network, `172.24.249.0/24`.

```
(undercloud) [stack@director ~]$ openstack subnet list \
-c 'Name' -c 'Subnet'
+---------------------+-----------------+
| Name                | Subnet          |
+---------------------+-----------------+
| ctlplane-subnet     | 172.25.249.0/24 |
...output omitted...
+---------------------+-----------------+
```

3.3. Use the **`openstack subnet show`** command to show the details of the `ctlplane-subnet` subnet. Notice the allocation pools IP addresses match the **`dhcp_start`** and **`dhcp_end`** parameter values set in the **`undercloud.conf`** file.

```
(undercloud) [stack@director ~]$ openstack subnet show ctlplane-subnet
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| allocation_pools  | 172.25.249.51-172.25.249.59 |
| cidr              | 172.25.249.0/24            |
| created_at        | 2018-09-26T00:38:29Z       |
| description       |                            |
| dns_nameservers   | 172.25.250.254             |
| enable_dhcp       | True                       |
| gateway_ip        | 172.25.249.200             |
...output omitted...
```

4. List all the registered bare-metal nodes on `director`. View the power management information for the `compute1` bare-metal node.

4.1. Use the **`openstack baremetal node list`** command to list all the registered bare-metal nodes.

```
(undercloud) [stack@director ~]$ openstack baremetal node list \
-c 'Name' -c 'Power State'
+-------------+-------------+
| Name        | Power State |
+-------------+-------------+
| controller0 | power on    |
| compute0    | power on    |
| computehci0 | power on    |
| compute1    | power on    |
| ceph0       | power on    |
+-------------+-------------+
```

4.2. Use the **`openstack baremetal node show`** command to view the power management information for the `compute1` node.

```
(undercloud) [stack@director ~]$ openstack baremetal node show \
-c 'driver' -c 'driver_info' compute1
+-------------+-----------------------------------------------------------------+
```

```
| Field       | Value                                                                |
+-------------+----------------------------------------------------------------+
| driver      | pxe_ipmitool                                                         |
| driver_info | {u'ipmi_port': u'623', u'ipmi_username': u'admin',                   |
|             | u'deploy_kernel': u'525385d3-7f95-4911-be6e-ea205db379a8',           |
|             | u'ipmi_address': u'172.25.249.112', u'deploy_ramdisk':               |
|             | u'246a9a84-f097-4ce1-82d0-e8a754ef94b2', u'ipmi_password':           |
|             | u'******'}                                                           |
+-------------+----------------------------------------------------------------+
```

Log off from `director`.

This concludes the guided exercise.

INTRODUCING CONTAINERIZED SERVICES

OBJECTIVES

After completing this section, students should be able to:

- Describe containerized services.
- Manage containers using basic commands.

CONTAINERIZED SERVICES INTRODUCTION

Unlike earlier versions of Red Hat OpenStack Platform, later versions use containers to run services. The containerized architecture ensures that each of the OpenStack services are isolated from the host. The main OpenStack services all use containers.

The containers are built using an image that is pulled from the Red Hat Customer Portal. Container service management is handled by the **`docker`** command, replacing the **`systemd`** commands. The underlying Linux system continues to use the **`systemd`** commands. The containers are updated by pulling new images and replacing the existing ones, and these containers are built using the `Kolla` tool set.

CONTAINER IMAGES AND REGISTRY

Red Hat OpenStack Platform 12 was the first release to implement a containerized control plane. Each service container runs an isolated stack without dependencies on other containers. Some of the key benefits of running containerized OpenStack services are described below:

Benefits of Using Containerized Control Plane Services

Stability
: Because each service container is isolated and not dependent on others, it reduces cascading failures in the event of a single service failure.

Security
: Containers images are immutable, by design. Containers that are compromised or improperly modified can be easily rebuilt by redeploying from the original image.

Life-cycle management
: Managing the life cycle of these service containers is easy, because they are isolated units. Because each service container can be upgraded and rolled back independently, you do not need to worry about breaking dependencies when upgrading or rolling back.

Service placement
: Administrators can move service containers between hosts as required.

Scalability
: Each service container can be scaled up or down quickly and easily. Currently, only one service container instance can be placed on a single node due to underlying dependencies on the host file system and networking.

Registry Methods Supported by Red Hat OpenStack Platform

Access to the images required to build containerized clouds requires that you either set up or use a pre-existing container registry. Red Hat OpenStack Platform supports pulling images from a container registry deployed on the undercloud or from an authorized remote registry. Using a

remote registry is a simple method for generating the initial configuration since the overcloud pulls the images from `https://registry.access.redhat.com` or the newer, authentication-required `https://registry.redhat.io`. However, using a remote registry can slow an installation because it pulls each image directly from the Red Hat Container Catalog.

A local registry configured on the undercloud is the recommended registry method. During the undercloud installation, an image registry is configured on port 8787. The deployment synchronizes the images from the remote registry and pushes them to the `docker-distribution` service.

```
(undercloud) [stack@director ~]$ systemctl status docker-distribution.service
● docker-distribution.service - v2 Registry server for Docker
   Loaded: loaded (/usr/lib/systemd/system/docker-distribution.service; enabled;
 vendor preset: disabled)
   Active: active (running) since Thu 2018-10-04 01:54:56 UTC; 2h 48min ago
 Main PID: 1497 (registry)
    Tasks: 8
   CGroup: /system.slice/docker-distribution.service
           └─1497 /usr/bin/registry serve /etc/docker-distribution/registry/
config.yml
```

The undercloud registry is populated with images required by the each of the services to be included in the overcloud. During the overcloud deployment, the images are pulled from this local registry. Using the local registry speeds up deployment and alleviates network congestion, but provides limited life-cycle management. Without manual intervention, images remain static for the life of the deployed overcloud.

Populating the Local Container Registry

By default the container images are pulled using the remote registry as defined in the **`/usr/share/openstack-tripleo-common-containers/container-images/overcloud_containers.yaml`** file. To use the undercloud as the local container registry the images are pulled from the remote registry by generating a template file with all the container images required for overcloud deployment. Use the following command to generate the template file:

```
(undercloud) [stack@director ~]$ openstack overcloud container image prepare \
-e /usr/share/openstack-tripleo-heat-templates/environments/ceph-ansible/ceph-
ansible.yaml \
--namespace=registry.access.redhat.com/rhosp13 \
--prefix=openstack- \
--tag-from-label {version}-{release} \
--output-images-file /home/stack/local_registry_images.yaml
```

The **`local_registry_images.yaml`** file is then used to upload the container images to the local container registry. Use the **`openstack overcloud container image upload`** command to upload the container images. Use the following command to list the container images uploaded to the local registry:

```
(undercloud) [stack@director ~]$ docker images --format="{{.Repository}}"
172.25.249.200:8787/rhosp13/openstack-neutron-server-ovn
172.25.249.200:8787/rhosp13/openstack-neutron-metadata-agent-ovn
172.25.249.200:8787/rhosp13/openstack-rabbitmq
172.25.249.200:8787/rhosp13/openstack-cinder-api
172.25.249.200:8787/rhosp13/openstack-ceilometer-central
```

```
172.25.249.200:8787/rhosp13/openstack-nova-placement-api
...output omitted...
```

CONTAINER COMMANDS

All of the container commands require `root` privileges.

Use the **`docker version`** command to find the version of Docker being used. Both the client and server versions are displayed.

```
[root@demo ~]# docker version
Client:
Version:         1.13.1
API version:     1.26
Package version: docker-1.13.1-63.git94f4240.el7.x86_64
Go version:      go1.9.2
Git commit:      94f4240/1.13.1
Built:           Mon Apr 30 15:45:42 2018
OS/Arch:         linux/amd64

Server:
Version:         1.13.1
API version:     1.26 (minimum version 1.12)
Package version: docker-1.13.1-63.git94f4240.el7.x86_64
Go version:      go1.9.2
Git commit:      94f4240/1.13.1
Built:           Mon Apr 30 15:45:42 2018
OS/Arch:         linux/amd64
Experimental:    false
```

The **`docker ps`** command lists all of the containers. Note that containers may exist on both the `controller` and `compute` nodes. You can use the **`grep`** command to refine the search, and you can use the **`--format`** option with the **`docker ps`** command to produce a more readable output. The **`-a`** option lists all containers. By default, only running containers are listed.

```
[root@demo ~]# docker ps
...output omitted...
4b7e5e4ddb73        172.25.249.200:8787/rhosp13/openstack-octavia-api:latest
         "kolla_start"            2 weeks ago         Up 7 days
                      octavia_api
3e35d2f06317        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
         "kolla_start"            2 weeks ago         Up 7 days
                      nova_metadata
c1c03b92e94c        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                      nova_api
70496141bc14        172.25.249.200:8787/rhosp13/openstack-octavia-
housekeeping:latest      "kolla_start"            2 weeks ago         Up 7 days
 (unhealthy)                              octavia_housekeeping
88730539bc82        172.25.249.200:8787/rhosp13/openstack-glance-api:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                      glance_api
...output omitted...
```

```
[root@demo ~]# docker ps | grep nova
3e35d2f06317        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
         "kolla_start"            2 weeks ago         Up 7 days
                     nova_metadata
c1c03b92e94c        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                     nova_api
75e0d6fdc452        172.25.249.200:8787/rhosp13/openstack-nova-scheduler:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                     nova_scheduler
148f680bbd6d        172.25.249.200:8787/rhosp13/openstack-nova-novncproxy:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                     nova_vnc_proxy
bf768ed3106c        172.25.249.200:8787/rhosp13/openstack-nova-consoleauth:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                     nova_consoleauth
5629fd35fd66        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
         "kolla_start"            2 weeks ago         Up 7 days
                     nova_api_cron
b5cad825ab04        172.25.249.200:8787/rhosp13/openstack-nova-conductor:latest
         "kolla_start"            2 weeks ago         Up 7 days (healthy)
                     nova_conductor
d6e78004e08c        172.25.249.200:8787/rhosp13/openstack-nova-placement-
api:latest        "kolla_start"            2 weeks ago         Up 7 days
                             nova_placement
```

```
[root@demo ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" | grep nova
nova_api_discover_hosts         Exited (0) 2 weeks ago
nova_metadata                   Up 7 days
nova_api                        Up 7 days (healthy)
nova_scheduler                  Up 7 days (healthy)
nova_vnc_proxy                  Up 7 days (healthy)
nova_consoleauth                Up 7 days (healthy)
nova_api_cron                   Up 7 days
nova_conductor                  Up 7 days (healthy)
nova_db_sync                    Exited (0) 2 weeks ago
nova_api_ensure_default_cell    Exited (0) 2 weeks ago
nova_api_map_cell0              Exited (0) 2 weeks ago
nova_placement                  Up 7 days
nova_api_db_sync                Exited (0) 2 weeks ago
nova_placement_init_log         Exited (0) 2 weeks ago
nova_metadata_init_log          Exited (0) 2 weeks ago
nova_api_init_logs              Exited (0) 2 weeks ago
```

NOTE

A status of **Exited** occurs when a particular service has started, carried out its necessary tasks, and then exited. It does not infer that the service has any kind of problem. Containers with the status of **Exited** are left in memory so that their log files and configuration can be accessed.

The **--filter** option is used to reduce the output from the **docker ps** command. There are many parameters which can be used with the **--filter** option. Example parameters include

name, **health**, and **status**. To more greatly manage the output, the **--filter** and **--format** options can be combined.

```
[root@demo ~]# docker ps --filter status=running --
format="{{.ID}}\t{{.Names}}\t{{.Status}}"
ced6f3501e8b    ovn-dbs-bundle-docker-0 Up 2 weeks
8a614232be08    openstack-manila-share-docker-0 Up 2 weeks
8b2ba32b7317    openstack-cinder-volume-docker-0    Up 2 weeks
d3ebb4e7779b    haproxy-bundle-docker-0 Up 2 weeks
...output omitted...
```

```
[root@demo ~]# docker ps --filter status=running --
format="{{.ID}}\t{{.Names}}\t{{.Status}}" | grep keystone
61e4f9938e2c    keystone_cron   Up 2 weeks
070c98d74131    keystone    Up 2 weeks (healthy)
```

The **docker stats** command shows the real-time statistics for a particular container's resource usage. The **stats** command can be compared to the **top** command on the Linux platform. You can pass a space-separated list of containers to the **docker stats** command to query multiple containers. Containers with a status of **Stopped** and **Exited** can be queried but will not return data.

```
[root@demo ~]# docker stats nova_conductor
CONTAINER      CPU %  MEM USAGE / LIMIT       MEM % NET I/O   BLOCK I/O
 PIDS
nova_conductor 0.04%  100.2 MiB / 7.639 GiB   1.28% 0 B / 0 B 23.5 MB / 4.1 kB  1
```

```
[root@demo ~]# docker stats nova_api keystone
CONTAINER     CPU %  MEM USAGE / LIMIT       MEM %  NET I/O   BLOCK I/O
 PIDS
nova_api      0.26%  165.9 MiB / 7.639 GiB   2.12%  0 B / 0 B 22.4 MB / 4.1 kB  14
keystone      0.20%  203.8 MiB / 7.639 GiB   2.61%  0 B / 0 B 25.3 MB / 4.1 kB  19
```

Use the **docker images** command to display the repository name, size, tags, image ID, and creation date for all available images. Add the ***image-name*** option to only display images from the specified repository. You can also use the **--format** option to customize the output. The **latest** tag is automatically attached to the highest version of each stored image.

```
[root@demo ~]# docker images
REPOSITORY                                                    TAG
  IMAGE ID            CREATED             SIZE
172.25.249.200:8787/rhosp13/openstack-neutron-server-ovn        latest
  80fa6c0a0a0a        5 weeks ago         867 MB
172.25.249.200:8787/rhosp13/openstack-cinder-api                latest
  1438961b75d3        5 weeks ago         992 MB
172.25.249.200:8787/rhosp13/openstack-ceilometer-central        latest
  6aab1d712c07        5 weeks ago         753 MB
```

```
[root@demo ~]# docker images 172.25.249.200:8787/rhceph/rhceph-3-rhel7
REPOSITORY                                  TAG     IMAGE ID     CREATED      SIZE
```

```
172.25.249.200:8787/rhceph/rhceph-3-rhel7   latest   fa3b551f0952 5 weeks ago   592
 MB
```

```
[root@demo ~]# docker images 172.25.249.200:8787/rhceph/rhceph-3-rhel7 \
--format="{{.Repository}}\t{{.Tag}}"
172.25.249.200:8787/rhceph/rhceph-3-rhel7   latest
```

The **`docker inspect`** command inspects the configuration of a container. The results are rendered in JSON. Any field can be extracted from the JSON output. Use the **`jq`** command to filter the JSON output.

```
[root@demo ~]# docker inspect cinder_api
[
  {
      "Id": "f08c...0b5c",
      "Created": "2018-08-01T16:14:59.026719599Z",
      "Path": "kolla_start",
      "Args": [],
      "State": {
                 "Status": "running",
                 "Running": true,
                 "Paused": false,
                 "Restarting": false,
                 "OOMKilled": false,
                 "Dead": false,
                 "Pid": 12936,
                 "ExitCode": 0,
                 "Error": "",
                 "StartedAt": "2018-08-13T06:29:19.129887492Z",
                 "FinishedAt": "2018-08-13T06:29:01.737937047Z"
                }
...output omitted...
```

```
[root@demo ~]# docker inspect cinder_api | jq .[0].Mounts
[
  {
      "Propagation": "rprivate",
      "RW": true,
      "Mode": "",
      "Destination": "/dev/log",
      "Source": "/dev/log",
      "Type": "bind"
  },
 ...output omitted...
```

To display the value of the `hosts path` instance parameter, use the **`--format`** option.

```
[root@demo ~]# docker inspect --format="{{.HostsPath}}" 61e4f9938e2c
/etc/hosts
```

The **`docker logs`** command shows the startup output from STDOUT. This command only works for containers that are started with the **`journald`** logging driver. The **`docker logs`** command

supports a range of options. The **`--since`** option shows container logs generated after a given date. The **`--tail`** option shows the number of rows specified from the end of the log file.

```
[root@demo ~]# docker logs cinder_api
+ sudo -E kolla_set_configs
INFO:__main__:Loading config file at /var/lib/kolla/config_files/config.json
INFO:__main__:Validating config file
INFO:__main__:Kolla config strategy set to: COPY_ALWAYS
INFO:__main__:Copying service configuration files
INFO:__main__:Deleting /etc/cinder/cinder.conf
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/cinder/cinder.conf to /
etc/cinder/cinder.conf
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/httpd/conf.d/10-
cinder_wsgi.conf to /etc/httpd/conf.d/10-cinder_wsgi.conf
INFO:__main__:Deleting /etc/httpd/conf.d/ssl.conf
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/httpd/conf.d/ssl.conf
 to /etc/httpd/conf.d/ssl.conf
```

NOTE

Containers remain in memory when they are stopped. This behavior allows an administrator to troubleshoot the container's log even after the container stops running.

To access a container, use the **`docker exec`** command to launch **`/bin/bash`**. This provides a bash prompt from which to run commands directly in the container itself. The container must already be up and running to execute further commands in it. A paused or stopped container will return an error. The command is executed in the container's default directory. Alternatively, you can run commands in the container non-interactively.

The **`docker exec`** command requires the **`-i`** or interactive option. The **`--detach`** option runs the command in the background. To set environment variables use the **`--env`** option. To run a string of commands together, use a shell construct, such as **`docker-compose exec sh -c '<command 1> && <command 2> && <command 3>'`**

```
[root@controller0 ~]# docker exec -it keystone /bin/bash
()[root@controller0 /]# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  openstack  opt  proc  root  run
  run_command  sbin  srv  sys  tmp  usr  var
```

```
[root@controller0 ~]# docker exec -t keystone /openstack/healthcheck
{"versions": {"values": [{"status": "stable", "updated": "2018-02-28T00:00:00Z",
 "media-types": [{"base": "application/json", "type": "application/
vnd.openstack.identity-v3+json"}], "id": "v3.10", "links": [{"href":
 "http://controller0.internalapi.overcloud.example.com:5000/v3/", "rel":
 "self"}]}, {"status": "deprecated", "updated": "2016-08-04T00:00:00Z",
 "media-types": [{"base": "application/json", "type": "application/
vnd.openstack.identity-v2.0+json"}], "id": "v2.0", "links": [{"href":
 "http://controller0.internalapi.overcloud.example.com:5000/v2.0/", "rel":
 "self"}, {"href": "https://docs.openstack.org/", "type": "text/html", "rel":
 "describedby"}]}]}}
300 172.24.1.1:5000 0.006 seconds
```

Using the **docker** command with the **stop**, **start**, and **restart** subcommands you can control the containers. Use the **docker ps** command to verify the status of the container.

```
[root@demo ~]# docker stop nova_api
nova_api
```

```
[root@demo ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" | grep nova_api
...output omitted...
nova_api    Exited (0) 30 seconds ago
...output omitted...
```

```
[root@demo ~]# docker start nova_api
nova_api
```

```
[root@demo ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" | grep nova_api
...output omitted...
nova_api    Up 1 second (health: starting)
...output omitted...
```

LOG FILES AND CONFIGURATION FILES

Log File Location

Previous versions of Red Hat OpenStack stored log files in **/var/log/*service-name***, for example **/var/log/cinder**. In the containerized architecture the log files are stored in **/var/log/containers/*service-name***, for example **/var/log/containers/cinder**.

NOTE

The **/var/log/*service*** log directories still exist in the containerized architecture, however those directories are empty except for a file called **readme.txt**.

```
[root@demo cinder]# cd /var/log/cinder
[root@demo cinder]# ll
total 4
-rw-r--r--. 1 root root 121 Aug 19 03:27 readme.txt
```

```
[root@demo cinder]# cat readme.txt
Log files from cinder containers can be found under
/var/log/containers/cinder and /var/log/containers/httpd/cinder-api.
```

```
[root@demo cinder]# cd /var/log/containers/cinder
[root@demo cinder]# ll
total 2084
-rw-r--r--. 1 42407 42407 1979355 Aug 21 07:02 cinder-api.log
-rw-r--r--. 1 42407 42407    8163 Aug 19 04:01 cinder-manage.log
-rw-r--r--. 1 42407 42407   13675 Aug 21 05:28 cinder-scheduler.log
-rw-r--r--. 1 42407 42407   11607 Aug 21 05:26 cinder-volume.log
```

Configuration File Location

Like log files, configuration files have also changed location. Previously they were stored in **/etc/*service***. In the containerized architecture, configuration is stored in files in the **/var/lib/config-data/puppet-generated/*service-name*** directory.

```
[root@demo ~]# ll /var/lib/config-data/puppet-generated/
...output omitted...
drwxr-xr-x. 3 root root 17 Aug 19 03:36 ceilometer
-rw-r--r--. 1 root root 33 Aug 19 03:36 ceilometer.md5sum
drwxr-xr-x. 4 root root 28 Aug 19 03:35 cinder
-rw-r--r--. 1 root root 33 Aug 19 03:35 cinder.md5sum
drwxr-xr-x. 3 root root 17 Aug 19 03:32 clustercheck
-rw-r--r--. 1 root root 33 Aug 19 03:32 clustercheck.md5sum
drwxr-xr-x. 4 root root 28 Aug 19 03:35 crond
-rw-r--r--. 1 root root 33 Aug 19 03:35 crond.md5sum
drwxr-xr-x. 3 root root 17 Aug 19 03:34 glance_api
...output omitted...
```

NOTE

The **/etc/*service*** directories still exist and contain the configuration files for that service. However, it is important to know that any change made in those configuration files will not change the configuration of the docker container. All container configuration changes must be made using the configuration files in **/var/lib/config-data/puppet-generated/service/etc/*service***.

```
[root@demo ~]# cd /etc/glance
[root@demo glance]# ll
total 424
-rw-r-----. 1 root glance 161914 Apr 24 18:17 glance-api.conf
-rw-r-----. 1 root glance  84916 Apr 24 18:17 glance-cache.conf
-rw-r-----. 1 root glance  83520 Apr 24 18:17 glance-registry.conf
-rw-r-----. 1 root glance  76733 Apr 24 18:17 glance-scrubber.conf
-rw-r--r--. 1 root glance    632 Jul 25 04:35 glance-swift.conf
drwxr-xr-x. 2 root root     4096 Jul 31 19:35 metadefs
-rw-r-----. 1 root glance   1388 Apr 24 18:17 policy.json
-rw-r-----. 1 root glance    941 Apr 24 18:17 rootwrap.conf
drwxr-xr-x. 2 root root       65 Jul 31 19:35 rootwrap.d
-rw-r-----. 1 root glance   1368 Apr 24 18:17 schema-image.json
```

```
[root@demo ~]# cd /var/lib/config-data/puppet-generated/glance_api/etc/glance/
[root@demo glance]# ll
total 244
-rw-r-----. 1 root 42415 162998 Aug 19 03:34 glance-api.conf
-rw-r-----. 1 root 42415  85034 Aug 19 03:34 glance-cache.conf
```

Only changes made in these configuration files are persistent over a restart of the service.

REFERENCES

For more information, refer to the *Transitioning to Containerized Services* section in the *Red Hat OpenStack Platform Guide* at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/transitioning_to_containerized_services/

VIEWING CONTAINERIZED SERVICE STRUCTURES

In this exercise, you will perform some of the most common administration tasks related to containerized services.

OUTCOMES

You should be able to:

- Perform common administration tasks related to containerized services.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab architecture-containerization setup`** command.

```
[student@workstation ~]$ lab architecture-containerization setup
```

1. Log in to `compute0` as the `heat-admin` user. Use the **`sudo -i`** command to change to the `root` user. Use the **`docker ps`** command to list the containers on the compute node. Log off from the `compute0` node, and then log in to the `controller0` node. Use the **`sudo -i`** command to change to the `root` user. On the `controller0` node run the **`docker ps`** command to list the containers. Note the difference between the two nodes.

   1.1. Log in to the `compute0` node as the `heat-admin` user. Use the **`docker ps`** command to list the containers. Log off from the `compute0` node.

```
[student@workstation ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]# docker ps
CONTAINER ID        IMAGE
            COMMAND             CREATED             STATUS                  PORTS
              NAMES
5873d9d3221c        172.25.249.200:8787/rhosp13/openstack-ovn-controller:latest
            "kolla_start"       3 days ago          Up 26 hours
              ovn_controller
f5de65054aa7        172.25.249.200:8787/rhosp13/openstack-neutron-metadata-agent-
ovn:latest   "kolla_start"       3 days ago          Up 26 hours (healthy)
               ovn_metadata_agent
...output omitted...
[root@compute0 ~]# exit
[heat-admin@compute0 ~]$ exit
```

   1.2. Log in to the `controller0` node as the `heat-admin` user. Use the **`sudo -i`** to become `root`. Use the **`docker ps`** command to list the containers.

```
[student@workstation ~]$ ssh heat-admin@controller0
```

```
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# docker ps
CONTAINER ID        IMAGE
         COMMAND                  CREATED             STATUS
   PORTS               NAMES
28fcdfb385fc        172.25.249.200:8787/rhosp13/openstack-ovn-northd:latest
         "/bin/bash /usr/lo..."   26 hours ago        Up 26 hours
                        ovn-dbs-bundle-docker-0
b8d5c72ffe23        172.25.249.200:8787/rhosp13/openstack-manila-share:pcmklatest
         "/bin/bash /usr/lo..."   26 hours ago        Up 26 hours
                        openstack-manila-share-docker-0
37fd300bb956        172.25.249.200:8787/rhosp13/openstack-cinder-volume:pcmklatest
         "/bin/bash /usr/lo..."   26 hours ago        Up 26 hours
                        openstack-cinder-volume-docker-0
...output omitted...
```

2. Use the **`docker images`** command to list the images used to create the service containers.

```
[root@controller0 ~]# docker images
REPOSITORY                                                          TAG
  IMAGE ID            CREATED             SIZE
172.25.249.200:8787/rhceph/rhceph-3-rhel7                           latest
  fa3b551f0952        2 weeks ago         592 MB
172.25.249.200:8787/rhosp13/openstack-neutron-server-ovn            latest
  80fa6c0a0a0a        5 weeks ago         867 MB
172.25.249.200:8787/rhosp13/openstack-cinder-api                    latest
  1438961b75d3        5 weeks ago         992 MB
...output omitted...
```

3. Use the **`docker inspect`** command to inspect the `cinder_api` container. Use the **`jq`** command to refine the search. You can refine the search to any section of the configuration file.

   3.1. Use the **`docker inspect`** command to inspect the `cinder_api` container.

```
[root@controller0 ~]# docker inspect cinder_api
[
 {
   "Id": "15454f4111f64e31970a38a748ba810117b218ed8a7b0b5b156113dceda4af5d",
   "Created": "2018-08-19T04:12:39.610382135Z",
   "Path": "kolla_start",
   "Args": [],
   "State": {
       "Status": "running",
       "Running": true,
       "Paused": false,
       "Restarting": false,
       "OOMKilled": false,
       "Dead": false,
       "Pid": 14058,
       "ExitCode": 0,
       "Error": "",
       "StartedAt": "2018-08-21T05:24:24.150488874Z",
       "FinishedAt": "2018-08-21T05:23:55.583984134Z"
   },
```

```
...output omitted...
```

3.2. Refine the search. Use the **jq** command to select the **Config** section. The **.[0].** reads the first element of the array in the response.

```
[root@controller0 ~]# docker inspect cinder_api | jq .[0].Config
{
  "StopSignal": "SIGTERM",
  "Labels": {
    "version": "13.0",
    "vendor": "Red Hat, Inc.",
    "vcs-type": "git",
    "vcs-ref": "1746128f0b1df74441d30bf0a1a6902fe244a591",
    "usage": "This image is very generic and does not serve a single use case. Use
 it as a base to build your own images.",
    "url": "https://access.redhat.com/containers/#/registry.access.redhat.com/
rhosp13/openstack-cinder-api/images/13.0-46",
    "summary": "Red Hat OpenStack Platform 13.0 cinder-api",
    "release": "46",
    "name": "rhosp13/openstack-cinder-api",
    "managed_by": "paunch",
    "config_id": "tripleo_step4",
...output omitted...
```

4. Use the **docker logs** command to inspect the keystone startup log files. The service management log files are stored in the **/var/log/containers/*service*** directory. Use the **less** command to inspect the swift log files.

4.1. Use the **docker logs** command to inspect the keystone startup log files.

```
[root@controller0 ~]# docker logs keystone
+ sudo -E kolla_set_configs
INFO:__main__:Loading config file at /var/lib/kolla/config_files/config.json
INFO:__main__:Validating config file
INFO:__main__:Kolla config strategy set to: COPY_ALWAYS
INFO:__main__:Copying service configuration files
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/httpd/conf.d/10-
keystone_wsgi_admin.conf to /etc/httpd/conf.d/10-keystone_wsgi_admin.conf
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/httpd/conf.d/10-
keystone_wsgi_main.conf to /etc/httpd/conf.d/10-keystone_wsgi_main.conf
INFO:__main__:Deleting /etc/httpd/conf.d/ssl.conf
...output omitted...
```

4.2. Use the **less** command to inspect the swift log files.

```
[root@controller0 ~]# view /var/log/containers/swift/swift.log
...output omitted...
stead. This option will be ignored in a future release.
Aug 19 00:13:08 controller0 object-server: Starting 1
Aug 19 00:13:08 controller0 object-server: Starting object replicator in daemon
 mode.
Aug 19 00:13:08 controller0 object-server: Starting object replication pass.
Aug 19 00:13:09 controller0 object-server: Nothing replicated for 0.0820679664612
 seconds.
```

```
Aug 19 00:13:09 controller0 object-server: Object replication complete. (0.00
 minutes)
Aug 19 00:13:11 controller0 account-server: Beginning replication run
Aug 19 00:13:11 controller0 account-server: Replication run OVER
Aug 19 00:13:11 controller0 account-server: Attempted to replicate 0 dbs in
 0.00213 seconds (0.00000/s)
Aug 19 00:13:11 controller0 account-server: Removed 0 dbs
...output omitted...
```

5. Use the **docker exec** command to determine the health of the OpenStack service in the `keystone` container.

```
[root@controller0 ~]# docker exec -t keystone /openstack/healthcheck
{"versions": {"values": [{"status": "stable", "updated": "2018-02-28T00:00:00Z",
 "media-types": [{"base": "application/json", "type": "application/
vnd.openstack.identity-v3+json"}], "id": "v3.10", "links": [{"href":
 "http://controller0.internalapi.overcloud.example.com:5000/v3/", "rel":
 "self"}]}, {"status": "deprecated", "updated": "2016-08-04T00:00:00Z",
 "media-types": [{"base": "application/json", "type": "application/
vnd.openstack.identity-v2.0+json"}], "id": "v2.0", "links": [{"href":
 "http://controller0.internalapi.overcloud.example.com:5000/v2.0/", "rel":
 "self"}, {"href": "https://docs.openstack.org/", "type": "text/html", "rel":
 "describedby"}]}]}}
300 172.24.1.1:5000 0.006 seconds
```

6. Use the **docker stop** command to stop the `nova_api` container. Use the **docker ps** command to check the status of the container. Use the **docker start** command to restart the `nova_api` container. Recheck the container status. Wait 40 to 50 seconds and recheck the container status. Ensure that the container status is *healthy*.

   6.1. Use the **docker stop** command to stop the `nova_api` container.

```
[root@controller0 ~]# docker stop nova_api
nova_api
```

   6.2. Use the **docker ps** command to check the status of the container.

```
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" \
| grep nova_api
...output omitted...
nova_api    Exited (0) 32 seconds ago
...output omitted...
```

   6.3. Use the **docker start** command to restart the `nova_api` container.

```
[root@controller0 ~]# docker start nova_api
nova_api
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" \
| grep nova_api
...output omitted...
nova_api    Up 2 seconds (health: starting)
```

```
...output omitted...
```

6.4. Recheck the container status. Wait 40 to 50 seconds and recheck the container status. Ensure that the container status is **healthy**.

```
[root@controller0 ~]# docker ps -a --format="{{.Names}}\t{{.Status}}" \
| grep nova_api
nova_api    Up About a minute (healthy)
```

6.5. Log out of `controller0`.

```
[root@controller0 ~]# logout
[student@workstation ~]$
```

Cleanup

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

```
[student@workstation ~]$ lab architecture-containerization cleanup
```

This concludes the guided exercise.

DESCRIBING THE OVERCLOUD

OBJECTIVES

After completing this section, students should be able to:

- Describe the core Red Hat OpenStack components on the overcloud.
- View the overcloud structure.

INTRODUCING THE OVERCLOUD

Overcloud nodes are deployed from the undercloud node using a dedicated, isolated provisioning network. When built, an overcloud is a production infrastructure with many interacting components. Red Hat OpenStack Director also performs activities such as updates and upgrades, scaling up and down, change management, and compliance. Therefore, the undercloud system is not intended to be uninstalled or decommissioned after the overcloud is installed. Currently, the undercloud can install a single overcloud with the stack name **`overcloud`**.

The overcloud usually consists of predefined roles, such as `Controller`, `Compute`, and different storage roles. Each of these default roles contains a set of services defined by the orchestration template used to deploy the overcloud. These roles are defined the **`/usr/share/openstack-tripleo-heat-templates/roles_data.yaml`** template file.

```
(undercloud) [stack@director ~]$ grep '^- name:' /usr/share/openstack-tripleo-
heat-templates/roles_data.yaml
- name: Controller
- name: Compute
- name: BlockStorage
- name: ObjectStorage
- name: CephStorage
```

Red Hat OpenStack Platform also provides a predefined custom role, named `ComputeHCI`, which deploys both the compute services and Ceph OSD services on the hyper-converged node. To use the `ComputeHCI` role, you need to generate a custom **`roles_data.yaml`** file that includes this role, along with all the other roles you are planning to use in your overcloud deployment.

OpenStack Core Services on the Controller Node

The overcloud controller nodes host the majority of the core services. The core services that are configured by default on the controller node are described below:

Identity Service (keystone)
: Provides user authentication and authorization to the OpenStack components running on the overcloud. The identity service supports multiple authentication mechanisms, including user name and password credentials, tokens, and other authentication protocols. This service is located on controller nodes.

OpenStack Networking Service (neutron)
: Provides the creation and management of a virtual networking infrastructure in an OpenStack cloud, including networks, subnets, routers, firewalls, and virtual private networks (VPNs). Designed as a pluggable architecture, OpenStack networking supports multiple vendors and networking technologies.

Dashboard (horizon)
Provides a modular, graphical user interface to manage OpenStack. It can launch server instances, configure networking topology, set role-based access controls, provision persistent and ephemeral storage, monitor runtime metrics, and organize projects and users.

Orchestration Service (heat)
Provides a template based methodology for creating and managing OpenStack cloud storage, networking and compute resources. A *Heat Orchestration Template (HOT)* defines a collection of resources, known as a *stack*, to be provisioned and deployed as a single, repeatable, running entity. In addition to recognizing essential resource types such as server instances, subnets, volumes, security groups, and floating IPs, templates provide additional configuration for advanced functionality, such as high availability, auto scaling, authentication, and nested stacks.

Compute Service (nova)
Provides and schedules on-demand virtual machines deployed and run on preconfigured compute nodes operating on nested virtualization or bare-metal hardware. The compute service on the controller node also provides the compute API service to serve API requests from the OpenStack clients.

Image Service (glance)
The image service, running on controller nodes, provides discovering, registering, and retrieving virtual machine (VM) images.

Block Storage (cinder)
The block storage service manages the storage volumes for virtual machine instances. This storage is persistent block storage. The block storage service, running on controller nodes, provides the block storage API and scheduling services.

Object Store (swift)
The object store 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. The actual object storage is located on an object store node, accessed by clients through the object storage proxy service.

Messaging (libraries for OpenStack)
The messaging service provides internal communication between several OpenStack services. The messaging service uses `RabbitMQ` as the back end. Most of the OpenStack services share common core libraries that provide a consistent, highly tested, and compatible feature set provided by `Oslo`. The `Oslo` libraries provide functionalities such as configuration, logging, caching, messaging, and database management. These libraries use the `olso.` prefix and generally have well-described library names, such as `oslo.messaging`, `oslo.policy`, and many others.

Additional Supported Services on Controller Nodes

Overcloud controller nodes also provide additional services:

Metering and Data Collection Service (ceilometer)
The data collection service 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 plug-ins to convert them into events and samples. The metering data is gathered by notification and polling agents. Polling agents run on all the overcloud nodes, poll the OpenStack infrastructure resources, such as the hypervisor, to publish meters on the notification bus. The notification agent runs on the controller node, listens to the notifications on the OpenStack notification bus and converts them into meter events and samples.

Metric Service (gnocchi)

The metric service is a time-series database used to store metrics and resources published by the data collection 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.

Load Balancing Service (octavia)

The load balancing service enables distribution of traffic to two or more instances so that the multitier architecture can manage a dynamic traffic load. 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. The service uses the OpenStack networking, compute, image, identity, and messaging services running on the controller and compute overcloud nodes.

Shared Filesystem 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.

Workflow Service (mistral)

The workflow service is used to define a set of tasks and relations in a complex multistep process called as workflow. A workflow is fault-tolerant so that if the task execution crashes at some point on one node then another active node of the system can automatically take on and continue from the same point where it stopped. The workflow service provides prebuilt workflows for common actions required on the overcloud, such as rotating Fernet keys.

Messaging Service (zaqar)

The messaging service provides the ability to send 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 the messaging service to surface events to end users and to communicate with guest agents that run in the overcloud.

OpenStack Core Services on Compute Nodes

The overcloud compute nodes host the core services required to launch an instance. The core services that are configured by default on the compute node are described below:

OpenStack Networking Service (neutron)

The networking service on a compute node runs the Open vSwitch agent which creates the node's virtual bridges. The service can also provide the metadata service for instances launched on the compute nodes.

Compute Service (nova)

The OpenStack compute service supports many hypervisors such as KVM, QEMU, and many others. The compute node runs a worker daemon named `nova-compute` that creates and terminates virtual machine instances through hypervisor APIs.

OpenStack Core Services on Compute HCI Nodes

The *Compute HCI (Hyper Converged Infrastructure)* node is a hyper-converged node that provides both compute and storage services by collocating the Ceph OSD and compute services. These nodes contain the Ceph OSD services and act as the actual storage. The node runs a compute daemon that is responsible for creating and terminating virtual machine instances through hypervisor APIs.

OpenStack Core Services on Ceph Storage Nodes

Red Hat Ceph Storage nodes provide storage for virtual machine instances, images, block storage, and ephemeral storage. These nodes contain the Ceph OSD services and act as the actual storage. Most Ceph Storage nodes use multiple disks. The Ceph monitor service runs on the controller node.

OpenStack Core Services on Block Storage Nodes

The block storage node consists of the `cinder-volume` service that manages the creation and deletion of block storage using the integrated back end. An integrated back end can be configured and deployed to the overcloud. Examples of integrated back ends include Red Hat Ceph Storage and single back-end configurations of Dell EqualLogic, Dell Storage Center, and NetApp appliances. The block storage scheduler service that schedules and routes requests to the appropriate node is hosted on the controller node. The block storage API service is also deployed on the controller node.

MANAGING THE OVERCLOUD USING THE UNDERCLOUD

The workflow service on the undercloud can manage multiple deployment plans and stacks. In a future release, the undercloud will be able to install, access, and manage multiple overclouds. Currently supported ongoing activities for the undercloud include:

- Monitoring the health of an overcloud.
- Gathering and storing metrics from an overcloud.
- Validating and introspecting new nodes for overcloud scaling.
- Performance testing of nodes, components, and scenarios.
- Performing minor release updates, such as security fixes.
- Performing automated major version upgrades to Red Hat OpenStack Platform.
- Auto scaling or replacing HA controllers, storage nodes, and compute nodes.
- Managing platform-level access to infrastructure nodes, including power management.

An overcloud deployment creates each of the registered nodes as one of the standard server roles or custom roles defined. To access an overcloud server over SSH, use the **`openstack server list`** command from the undercloud and use the undercloud admin's credentials to find the IP address of the desired node:

```
(undercloud) [stack@director ~]$ openstack server list \
-c 'Name' -c 'Status' -c 'Networks'
+-------------+--------+------------------------+
| Name        | Status | Networks               |
+-------------+--------+------------------------+
| controller0 | ACTIVE | ctlplane=172.25.249.53 |
| ceph0       | ACTIVE | ctlplane=172.25.249.57 |
| compute0    | ACTIVE | ctlplane=172.25.249.55 |
| compute1    | ACTIVE | ctlplane=172.25.249.52 |
| computehci0 | ACTIVE | ctlplane=172.25.249.56 |
+-------------+--------+------------------------+
```

To browse those servers, use the `heat-admin` user, the same Linux user account used by the overcloud deployment process to access and configure the systems using SSH.

```
(undercloud) [stack@director ~]$ ssh heat-admin@172.25.249.53
[heat-admin@controller0 ~]$
```

In a high-availability overcloud deployment, the overcloud is comprised of multiple controller nodes in a cluster. RHOSP Director installs a duplicate set of OpenStack components on each controller node and manages them as a single service. Red Hat OpenStack Platform uses Pacemaker as a cluster resource manager, HAProxy for load balancing cluster services, and MariaDB Galera for database replication. Each resource managed by Pacemaker sets a virtual IP address that the OpenStack clients use to request access to a service. If the controller node that is assigned to that IP address fails, the IP address is reassigned to a different controller.

In the classroom environment, the overcloud is deployed with one controller node in the cluster. You can use the **`pcs status`** command to list the general Pacemaker information, virtual IP addresses, services, and other Pacemaker information.

```
[heat-admin@controller0 ~]$ sudo pcs status
(1)Cluster name: tripleo_cluster
Stack: corosync
Current DC: controller0 (version 1.1.18-11.el7_5.3-2b07d5c5a9) - partition with
 quorum
Last updated: Mon Oct  8 09:21:04 2018
Last change: Sat Oct  6 08:57:54 2018 by root via cibadmin on controller0

5 nodes configured
21 resources configured

Online: [ controller0 ]
GuestOnline: [ galera-bundle-0@controller0 ovn-dbs-bundle-0@controller0 rabbitmq-
bundle-0@controller0 redis-bundle-0@controller0 ]

Full list of resources:

 (2)Docker container: rabbitmq-bundle [172.25.249.200:8787/rhosp13/openstack-
rabbitmq:pcmklatest]
   rabbitmq-bundle-0  (ocf::heartbeat:rabbitmq-cluster):  Started controller0
 (3)Docker container: haproxy-bundle [172.25.249.200:8787/rhosp13/openstack-
haproxy:pcmklatest]
   haproxy-bundle-docker-0  (ocf::heartbeat:docker):  Started controller0
 ...
 ip-172.25.249.50 (ocf::heartbeat:IPaddr2): Started controller0
 (4)ip-172.25.250.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.1.51 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.1.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.3.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.4.50 (ocf::heartbeat:IPaddr2): Started controller0
 ...output omitted...
```

1. Cluster name
2. 3. Services managed by Pacemaker
4. Public virtual IP address

Accessing Services on the Overcloud

When the overcloud installation completes it creates the **`overcloudrc`** identity environment file under the user's home directory with the admin credentials to access the overcloud. You can source the **`overcloudrc`** file to interact with the overcloud services using the OpenStack CLI client.

In the classroom environment, the OS_AUTH_URL environment variable in the **overcloudrc** file is set to the public endpoint of the identity service running on the overcloud. The identity service public endpoint is the virtual IP address configured to access the service. The classroom environment uses non-SSL for all public endpoints for the OpenStack services on the overcloud.

```
(undercloud) [stack@director ~]$ source ~/overcloudrc
(overcloud) [stack@director ~]$ openstack endpoint list \
-c 'Service Type' -c 'Interface' -c 'URL'
+----------------+-----------+---------------------------+
| Service Type   | Interface | URL                       |
+----------------+-----------+---------------------------+
...output omitted...
| identity       | public    | http://172.25.250.50:5000 |
+----------------+-----------+---------------------------+
(overcloud) [stack@director ~]$ env | grep ^OS_*
OS_USER_DOMAIN_NAME=Default
OS_IMAGE_API_VERSION=2
OS_PROJECT_NAME=admin
OS_IDENTITY_API_VERSION=3
OS_PASSWORD=redhat
OS_AUTH_TYPE=password
OS_AUTH_URL=http://172.25.250.50:5000//v3
OS_USERNAME=admin
OS_VOLUME_API_VERSION=3
OS_NO_CACHE=True
OS_CLOUDNAME=overcloud
OS_PROJECT_DOMAIN_NAME=Default
```

VIEWING THE NETWORKING SERVICE ON THE OVERCLOUD

Red Hat OpenStack Platform Director uses the provisioning network to deploy new nodes using PXE boot and to orchestrate the overcloud installation on the bare-metal servers. In a default overcloud deployment the services are assigned to the configured provisioning network. However, the Director can divide overcloud network traffic into several isolated networks. These isolated networks can either be provisioned to use different network interfaces or can be isolated using VLANs. In the classroom environment, the public network is provisioned on a separate network interface whereas the internal overcloud network uses VLANs for isolation. The provisioning network is used for the power management of nodes. The network traffic types in the classroom environment are described below:

Network Type Assignment

NETWORK	PURPOSE	INTERFACE	NODE ROLE
Provisioning	For secure installation and virtual power management of overcloud nodes.	`eth0`	All nodes
Internal API	For private communication between OpenStack services across overcloud nodes.	`eth1` (`vlan10`)	Controller, Compute, ComputeHCI
Tenant	For self-service tenant network traffic across overcloud nodes, segregated using VLANs or tunnels layered on this VLAN.	`eth1` (`vlan20`)	Controller, Compute, ComputeHCI

NETWORK	PURPOSE	INTERFACE	NODE ROLE
Storage	For data plane traffic between data clients and the Ceph storage server.	`eth1` (`vlan30`)	All nodes
Storage management	For control plane traffic between service clients and the Ceph storage server.	`eth1` (`vlan40`)	Controller, ComputeHCI, CephStorage
Management	For optional enterprise management services, private to the overcloud.	`eth1` (`vlan50`)	All nodes
External	Configured as the gateway to public access.	`eth2`	All nodes

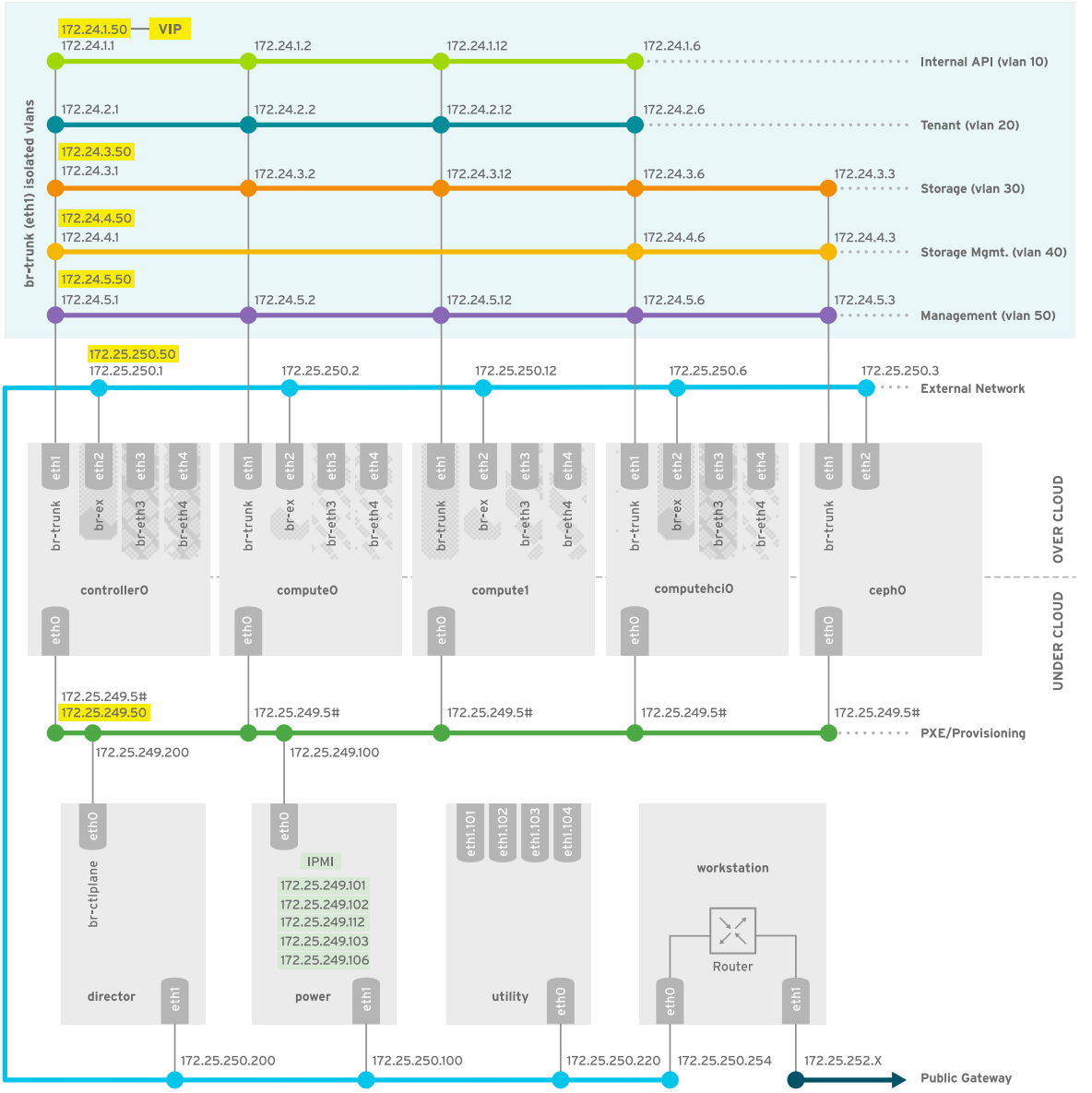


Figure 1.3: Completed classroom overcloud structure

View the overcloud's configured network interfaces. The `eth0` interface is the 172.25.249.0 provisioning network; the `br-ex` bridge is the 172.25.250.0 public network.

```
[heat-admin@controller0 ~]$ ip addr | grep 'eth0\|br-ex'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    inet 172.25.249.53/24 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 UNKNOWN
 group default qlen 1000
    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
```

The `br-trunk` bridge contains the isolated VLANs for the internal OpenStack network traffic.

```
[heat-admin@controller0 ~]$ sudo ovs-vsctl show
...output omitted...
    Bridge br-trunk
        fail_mode: standalone
        Port "vlan40"
            tag: 40
            Interface "vlan40"
                type: internal
        Port "vlan50"
            tag: 50
            Interface "vlan50"
                type: internal
        Port "vlan10"
            tag: 10
            Interface "vlan10"
                type: internal
        Port br-trunk
            Interface br-trunk
                type: internal
        Port "eth1"
            Interface "eth1"
        Port "vlan20"
            tag: 20
            Interface "vlan20"
                type: internal
        Port "vlan30"
            tag: 30
            Interface "vlan30"
                type: internal
    Bridge br-ex
        fail_mode: standalone
        Port br-ex
            Interface br-ex
                type: internal
        Port "eth2"
            Interface "eth2"
    ...output omitted...
```

Run the following commands to make sure that the networks are active and that there are services actively listening on the IP addresses. In the output below, the **`ip`** command shows the IP addresses on which the storage VLAN (`vlan30`) interface is listening. The **`netstat`** command displays the processes listening on the storage VLAN interface. The processes listening on the

storage VLAN interface are: `Ceph monitor`, `Ceph MDS`, and `Ceph manager`, and also has an `ntp` process listening on port 123.

```
[heat-admin@controller0 ~]$ ip addr show vlan30
17: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
 group default qlen 1000
    link/ether fe:45:09:f6:23:05 brd ff:ff:ff:ff:ff:ff
    inet 172.24.3.1/24 brd 172.24.3.255 scope global vlan30
       valid_lft forever preferred_lft forever
    inet 172.24.3.50/32 brd 172.24.3.255 scope global vlan30
       valid_lft forever preferred_lft forever
    inet6 fe80::fc45:9ff:fef6:2305/64 scope link
       valid_lft forever preferred_lft forever
[heat-admin@controller0 ~]$ sudo netstat -tupln | grep 172.24.3.1
tcp   0  0 172.24.3.1:6789   0.0.0.0:*   LISTEN  7479/ceph-mon
tcp   0  0 172.24.3.1:6800   0.0.0.0:*   LISTEN  6890/ceph-mds
tcp   0  0 172.24.3.1:8080   0.0.0.0:*   LISTEN  5752/python2
tcp   0  0 172.24.3.1:6801   0.0.0.0:*   LISTEN  6988/ceph-mgr
udp   0  0 172.24.3.1:123    0.0.0.0:*           700/ntpd
```

UNDERSTANDING OPENSTACK CONTAINERIZED SERVICES

In Red Hat OpenStack Platform 13 control plane services run in containers that provide methods to keep each service within its own isolated namespace and are not dependent on each other. Red Hat provides a set of prebuilt container images through the Red Hat Container Catalog. The container images which are pulled from the Red Hat Customer Portal and stored in the image repository on the Director node.

Working with OpenStack Service Containers

`Kolla` is a project aimed at deploying OpenStack in containers. It consists of container images and recipes for each of the OpenStack services as well as multiple options for orchestrating the containerized services. Red Hat OpenStack Platform uses `Kolla` container recipes to build most container images and deploy them using Red Hat OpenStack Platform Director.

In Red Hat OpenStack Platform 13, all of the OpenStack services run inside containers. Use the following command to list the service containers, the container image used by the containers, and their status on the controller node:

```
[root@controller0 ~]# docker ps \
--format "table {{.Names}}\t{{.Image}}\t{{.Status}}"
...output omitted...
nova_metadata  172.25.249.200:8787/rhosp13/openstack-nova-api:latest       Up 4
 hours
nova_api       172.25.249.200:8787/rhosp13/openstack-nova-api:latest       Up 4
 hours (healthy)
glance_api     172.25.249.200:8787/rhosp13/openstack-glance-api:latest     Up 4
 hours (healthy)
octavia_worker 172.25.249.200:8787/rhosp13/openstack-octavia-worker:latest Up 4
 hours (healthy)
ovn_controller 172.25.249.200:8787/rhosp13/openstack-ovn-controller:latest Up 4
 hours
aodh_listener  172.25.249.200:8787/rhosp13/openstack-aodh-listener:latest  Up 4
 hours (healthy)
```

```
...output omitted...
```

Use the following command to inspect the container host configuration of a containerized service. The following output lists the directories where the container locates the configuration files and where the container stores the logs. Further looking at the container host configuration, the **/var/lib/kolla/config_files/src** directory on the containerized service is mapped to **/var/lib/config-data/puppet-generated/nova/** on the host operating system. Similarly, the **/var/log/nova** directory on the container is mapped to **/var/log/containers/nova** on the host operating system.

```
[root@controller0 ~]# docker inspect nova_scheduler \
--format "{{json .HostConfig.Binds}}" | jq .
[
  "/etc/ssh/ssh_known_hosts:/etc/ssh/ssh_known_hosts:ro",
  "/run:/run",
  "/etc/pki/ca-trust/extracted:/etc/pki/ca-trust/extracted:ro",
  "/etc/pki/tls/certs/ca-bundle.trust.crt:/etc/pki/tls/certs/ca-
bundle.trust.crt:ro",
  "/etc/pki/tls/certs/ca-bundle.crt:/etc/pki/tls/certs/ca-bundle.crt:ro",
  "/etc/pki/tls/cert.pem:/etc/pki/tls/cert.pem:ro",
  "/dev/log:/dev/log",
  "/etc/puppet:/etc/puppet:ro",
  "/var/log/containers/nova:/var/log/nova",
  "/var/lib/kolla/config_files/nova_scheduler.json:/var/lib/kolla/config_files/
config.json:ro",
  "/etc/hosts:/etc/hosts:ro",
  "/etc/localtime:/etc/localtime:ro",
  "/var/lib/config-data/puppet-generated/nova/:/var/lib/kolla/config_files/src:ro"
]
```

Configuration Management of Containerized Services

The overcloud deployment process uses the orchestration templates as input to inject the Puppet-defined configuration changes into the container images. The **docker-puppet.py** script manages the configuration file generation and bootstrapping of each containerized service deployed on the overcloud. The tool is present in the **/usr/share/openstack-tripleo-heat-templates/docker/docker-puppet.py** directory on the Director node. The tool creates an initial container with the service binaries and Puppet installed. It then writes the service configurations to the host operating system during the overcloud deployment. While bootstrapping the service container being initialized and configured, an additional container is started to perform the tasks required to interact with the service container, such as creating identity service endpoints and the users required to launch the service container. The limitation of the **docker-puppet.py** script is that it can only be used with Puppet and is tightly integrated with the orchestration services.

With the **paunch** utility you can also launch and manage containers using the configuration data found in the TripleO service templates. This can also be used for bootstrapping a containerized OpenStack service. The **paunch** utility can also be used outside of TripleO for launching and performing configuration management of containers.

The **paunch** utility uses YAML configuration data to launch and manage containers. Operations are executed by **paunch** using the **docker**command. A container that is not created by **paunch** cannot be altered using the **paunch** command. The utility is available via the paunch CLI or by installing the *python* package.

The **paunch** utility sets labels on running containers. **paunch** requires that containers have a label to determine which containers to perform operation on. If a label has not been set then **paunch** does not know the state of the container.

The **paunch** utility uses idempotent behavior. Idempotency behavior leaves containers running when the configuration is not changed. If the configuration of a container changes then the entire container is replaced. Idempotency behavior means that the same **`--config-id`** can be used even when the container's configuration is modified.

Changes made directly inside the container are not persistent over a restart. To ensure that a change is persistent the relevant parameter inside the configuration is edited. The **paunch** utility reads the configuration files to recreate the container.

All the non-HA containerized services are managed by the **paunch** utility. The HA service containers such HAProxy, Galera, and RabbitMQ are created and managed by Pacemaker. Use the following command to determine if the specific containerized service is managed by the **paunch** utility:

```
[root@controller0 ~]# docker inspect nova_api -f "{{.Config.Labels.managed_by}}"
paunch
```

The **paunch debug** command allows you to perform specific actions on a given container. This can be used to launch a container with a custom configuration file, dump the configuration file of a specific container, or to get the command used to start the container. In the output below, the **paunch** utility is used to get the command used to start the `nova_scheduler` container.

```
[root@controller0 ~]# paunch debug --file \
/var/lib/tripleo-config/hashed-docker-container-startup-config-step_4.json \
--container nova_scheduler --action print-cmd
docker run --name nova_scheduler-3q0d0tt3 --
detach=true --env=KOLLA_CONFIG_STRATEGY=COPY_ALWAYS --
env=TRIPLEO_CONFIG_HASH=4dc7b008673f1e2565e5719b4002e8bf --net=host --health-
cmd=/openstack/healthcheck --privileged=false --restart=always --volume=/etc/
hosts:/etc/hosts:ro --volume=/etc/localtime:/etc/localtime:ro --volume=/etc/
pki/ca-trust/extracted:/etc/pki/ca-trust/extracted:ro --volume=/etc/pki/tls/
certs/ca-bundle.crt:/etc/pki/tls/certs/ca-bundle.crt:ro --volume=/etc/pki/tls/
certs/ca-bundle.trust.crt:/etc/pki/tls/certs/ca-bundle.trust.crt:ro --volume=/
etc/pki/tls/cert.pem:/etc/pki/tls/cert.pem:ro --volume=/dev/log:/dev/log --
volume=/etc/ssh/ssh_known_hosts:/etc/ssh/ssh_known_hosts:ro --volume=/etc/
puppet:/etc/puppet:ro --volume=/var/log/containers/nova:/var/log/nova --volume=/
var/lib/kolla/config_files/nova_scheduler.json:/var/lib/kolla/config_files/
config.json:ro --volume=/var/lib/config-data/puppet-generated/nova/:/var/
lib/kolla/config_files/src:ro --volume=/run:/run 172.25.249.200:8787/rhosp13/
openstack-nova-scheduler:latest
```

Managing Containerized Service Configuration

The configuration files for containerized services are stored in **`/var/lib/config-data/puppet-generated/`*`service`*`/etc/`*`service`***, for example, **`/var/log/config-data/puppet-generated/keystone/etc/keystone`**. The exception to that rule is the `aodh` container, for which the configuration files are found in the **`/var/lib/config-data/puppet-generated/aodh/`** directory.

A common administration trick is to modify the configuration inside the container and then use the `SIGHUP` signal to reload the configuration changes. These changes are ephemeral and on restarting the container the original configurations are restored.

To make persistent changes to the configuration, it must be done inside the bind mount directory of the container. The following commands show how to make changes to configuration to make it persistent in the **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** file.

```
[root@controller0 ~]# docker exec -u root keystone crudini --get /etc/keystone/
keystone.conf DEFAULT debug
False
[root@controller0 ~]# crudini --get /var/lib/config-data/puppet-generated/
keystone/etc/keystone/keystone.conf DEFAULT debug
False
[root@controller0 ~]# crudini --set /var/lib/config-data/puppet-generated/
keystone/etc/keystone/keystone.conf DEFAULT debug True
[root@controller0 ~]# docker restart keystone
keystone
[root@controller0 ~]# docker exec -u root keystone crudini --get /etc/keystone/
keystone.conf DEFAULT debug
True
```

Viewing Containerized Service Log Files

The containerized service log files are stored in **/var/log/containers/*service***, for example **/var/log/containers/cinder**. Containers also retain memory-based log structures that store the container's console STDOUT activity. Use **docker logs *service*** to view the container's console activity, for example **docker logs keystone**.

VIEWING STORAGE PROVIDER NODES

Integrating Red Hat OpenStack Platform with Red Hat Ceph Storage provides a highly scalable and redundant object, block, and file storage solution. The new shared file system service interfaces with the CephFS driver, providing a unified storage platform. Red Hat OpenStack Platform now supports the consolidation of compute and storage functions to reduce the overall number of servers needed for a given workload, increasing efficiency and cutting costs.

Hyper-converged infrastructure is becoming increasingly common for private cloud and network functions virtualization (NFV) infrastructure core and edge workloads. These platforms offer collocated, software-defined compute and storage resources, as well as a unified management interface, all running on economical, industry-standard hardware.

Viewing Ceph Services on the Controller Node

The Ceph storage service is now containerized. All Ceph commands which used to run on the Ceph Storage node now run on the controller node. List the Ceph containers on the `controller0` node.

```
[root@controller0 ~]# docker ps --format "{{.Names}}" | grep ceph
ceph-mgr-controller0
ceph-mds-controller0
ceph-mon-controller0
```

Ceph containers use `systemd` scripts to manage the containers. In the following output you can see the actual monitor service is named `ceph-mon@controller0.service`:

```
[root@controller0 ~]# systemctl list-units | grep ceph
ceph-mds@controller0.service    loaded active running   Ceph MDS
ceph-mgr@controller0.service    loaded active running   Ceph Manager
ceph-mon@controller0.service    loaded active running   Ceph Monitor
```

```
[root@controller0 ~]# systemctl status ceph-mon@controller0.service
● ceph-mon@controller0.service - Ceph Monitor
   Loaded: loaded (/etc/systemd/system/ceph-mon@.service; enabled; vendor preset:
 disabled)
   Active: active (running) since Mon 2018-10-08 08:52:00 UTC; 11h ago
  Process: 14616 ExecStartPre=/usr/bin/docker rm ceph-mon-%i (code=exited,
 status=1/FAILURE)
 Main PID: 14659 (docker-current)
   CGroup: /system.slice/system-ceph\x2dmon.slice/ceph-mon@controller0.service
           └─14659 /usr/bin/docker-current run --rm --name ceph-mon-
controller0 ...
```

Use the **`systemctl`** command to restart the Ceph containers on the controller node.

To list the OSDs, use the **`ceph osd ls`** command from the controller node.

```
[heat-admin@controller0 ~]$ ceph osd ls
0
1
2
3
4
5
```

To verify the status of the cluster, use the **`ceph status`** command.

```
[heat-admin@controller0 ~]$ ceph status
  cluster:
    id:     945a02d4-c931-11e8-b4cd-52540001fac8
    health: HEALTH_OK

  services:
    mon: 1 daemons, quorum controller0
    mgr: controller0(active)
    mds: cephfs-1/1/1 up  {0=controller0=up:active}
    osd: 6 osds: 6 up, 6 in

  data:
    pools:   7 pools, 416 pgs
    objects: 744 objects, 5617 MB
    usage:   6270 MB used, 107 GB / 113 GB avail
    pgs:     416 active+clean
```

To check the cluster's data storage usage, use the **`ceph df`** command.

```
[heat-admin@controller0 ~]$ ceph df
GLOBAL:
    SIZE     AVAIL     RAW USED     %RAW USED
    113G      107G        6270M          5.37
POOLS:
    NAME                ID     USED      %USED     MAX AVAIL     OBJECTS
    images              1      5617M      5.28       100774M         723
    metrics             2          0         0       100774M           0
    backups             3          0         0       100774M           0
```

```
    vms                 4          0         0       100774M           0
    volumes             5          0         0       100774M           0
    manila_data         6          0         0       100774M           0
    manila_metadata     7       3189         0       100774M          21
```

Viewing Ceph Services on the Ceph Storage and Compute HCI Nodes

The Ceph storage and Compute HCI nodes contain the OSDs. List the Ceph containers on the `ceph0` and `computehci0` nodes.

```
[root@ceph0 ~]# docker ps --format '{{ .Names}}' | grep ceph
ceph-osd-ceph0-vdc
ceph-osd-ceph0-vdb
ceph-osd-ceph0-vdd
```

```
[root@computehci0 ~]# docker ps --format '{{ .Names}}' | grep ceph
ceph-osd-computehci0-vdb
ceph-osd-computehci0-vdd
ceph-osd-computehci0-vdc
```

REFERENCES

Leveraging Containers and OpenStack
https://www.openstack.org/containers/leveraging-containers-and-openstack

Tips and Tricks for containerizing services
https://docs.openstack.org/tripleo-docs/latest/install/containers_deployment/tips_tricks.html

Further information is available in the *Deploying an Overcloud with Containerized Red Hat Ceph* for Red Hat OpenStack Platform 13; at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/deploying_an_overcloud_with_containerized_red_hat_ceph/

VIEWING THE OVERCLOUD ARCHITECTURE

In this exercise, you will view the architecture overcloud on your classroom system. You will verify the operation and configuration of the overcloud nodes.

OUTCOMES

You should be able to:

- Connect to and observe the undercloud virtual machine.
- Connect to and observe the overcloud virtual machines.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab architecture-overcloud setup`** command.

```
[student@workstation ~]$ lab architecture-overcloud setup
```

1. Log in to `director` as the `stack` user. Observe that the **`stackrc`** environment file automatically loaded. You will use the `stack` user's authentication environment to query and manage the undercloud.

   1.1. Log in to `director` as the `stack` user. No password is required. View the `stack` user's environment, which is used to connect to the undercloud.

```
[student@workstation ~]$ ssh stack@director
(undercloud) [stack@director ~]$ env | grep OS_
OS_BAREMETAL_API_VERSION=1.34
OS_USER_DOMAIN_NAME=Default
OS_PROJECT_NAME=admin
OS_IDENTITY_API_VERSION=3
OS_PASSWORD=redhat
OS_AUTH_TYPE=password
PS1=${OS_CLOUDNAME:+($OS_CLOUDNAME)} [\u@\h \W]\$
OS_AUTH_URL=https://172.25.249.201:13000/
OS_USERNAME=admin
OS_NO_CACHE=True
OS_CLOUDNAME=undercloud
OS_PROJECT_DOMAIN_NAME=Default
```

   1.2. View the current overcloud server list to find the provisioning network address for each node. The IP addresses shown here may differ from yours. Log out from the `director` node.

```
(undercloud) [stack@director ~]$ openstack server list \
```

```
-c Name -c Status -c Networks
+-------------+--------+-----------------------+
| Name        | Status | Networks              |
+-------------+--------+-----------------------+
| controller0 | ACTIVE | ctlplane=172.25.249.59 |
| compute1    | ACTIVE | ctlplane=172.25.249.56 |
| ceph0       | ACTIVE | ctlplane=172.25.249.54 |
| compute0    | ACTIVE | ctlplane=172.25.249.53 |
| computehci0 | ACTIVE | ctlplane=172.25.249.58 |
+-------------+--------+-----------------------+
```

```
(undercloud) [stack@director ~]$ exit
logout
Connection to director closed.
[student@workstation ~ ]$
```

2. Use the `heat-admin` account to log in to the `controller0` node to view the unique services running on it. The `heat-admin` account is configured with the SSH keys for the `stack` user from `director` to allow passwordless access.

   2.1. Log in to the `controller0` node as the `heat-admin` user. Use the **`ip addr`** command to list the network configuration.

```
(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 UP
    inet 172.25.249.53/24 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
14: 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
15: 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
16: 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
17: 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
18: 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
```

   2.2. Use the **`ovs-vsctl`** command to list the bridges on the `controller0` node.

```
[heat-admin@controller0 ~]$ sudo ovs-vsctl list-br
br-eth3
br-eth4
br-ex
br-int
```

```
br-trunk
```

2.3. Use the **ovs-vsctl** command to list the interfaces on br-trunk.

```
[heat-admin@controller0 ~]$ sudo ovs-vsctl list-ifaces br-trunk
eth1
vlan10
vlan20
vlan30
vlan40
vlan50
```

2.4. Use the **ovs-vsctl** command to list the interfaces on br-ex.

```
[heat-admin@controller0 ~]$ sudo ovs-vsctl list-ifaces br-ex
eth2
...output omitted...
```

2.5. Use the **docker** command to list the containers on the controller0 node. Log off from the controller0 node.

```
[heat-admin@controller0 ~]$ sudo docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                              STATUS
ovn-dbs-bundle-docker-0            Up 7 hours
openstack-manila-share-docker-0    Up 7 hours
openstack-cinder-volume-docker-0   Up 7 hours
haproxy-bundle-docker-0            Up 7 hours
redis-bundle-docker-0              Up 7 hours
galera-bundle-docker-0             Up 7 hours
rabbitmq-bundle-docker-0           Up 7 hours
...output omitted...
[heat-admin@controller0 ~]$ exit
(undercloud) [stack@director ~]$
```

3. Log in to the compute0 node as the head-admin user. List the relevant services and network configuration, and then log off.

3.1. Log in to the compute0 node as the heat-admin user. Use the **ip addr** command to list the network configuration.

```
(undercloud) [stack@director ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ ip addr | grep -E 'eth0|vlan|eth2'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    inet 172.25.249.53/24 brd 172.25.249.255 scope global eth0
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP
14: vlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.1.2/24 brd 172.24.1.255 scope global vlan10
15: vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.2.2/24 brd 172.24.2.255 scope global vlan20
16: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.3.2/24 brd 172.24.3.255 scope global vlan30
17: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
```

```
    inet 172.24.5.2/24 brd 172.24.5.255 scope global vlan50
```

3.2. Use the **ovs-vsctl** command to list the bridges on the `compute0` node.

```
[heat-admin@compute0 ~]$ sudo ovs-vsctl list-br
br-eth3
br-eth4
br-ex
br-int
br-trunk
```

3.3. Use the **ovs-vsctl** command to list the interfaces on `br-trunk`.

```
[heat-admin@compute0 ~]$ sudo ovs-vsctl list-ifaces br-trunk
eth1
vlan10
vlan20
vlan30
vlan50
```

3.4. Use the **docker** command to list the containers on the `compute0` node. Log off from the `compute0` node.

```
[heat-admin@compute0 ~]$ sudo docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                       STATUS
ovn_metadata_agent          Up 7 hours (healthy)
ovn_controller              Up 7 hours
nova_compute                Up 7 hours (healthy)
logrotate_crond             Up 7 hours
ceilometer_agent_compute    Up 7 hours
nova_migration_target       Up 7 hours
iscsid                      Up 7 hours
nova_libvirt                Up 7 hours
nova_virtlogd               Up 7 hours
[heat-admin@compute0 ~]$ exit
(undercloud) [stack@director ~]$
```

4. Log in to the `computehci0` node as the `head-admin` user. List the relevant services and network configuration, and then log off.

4.1. Log in to the `computehci0` node as the `heat-admin` user. Use the **ip addr** command to list the network configuration.

```
(undercloud) [stack@director ~]$ ssh heat-admin@computehci0
[heat-admin@computehci0 ~]$ ip addr | grep -E 'eth0|vlan|eth2'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    inet 172.25.249.56/24 brd 172.25.249.255 scope global eth0
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP
14: vlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.1.6/24 brd 172.24.1.255 scope global vlan10
15: vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.2.6/24 brd 172.24.2.255 scope global vlan20
16: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
```

```
    inet 172.24.3.6/24 brd 172.24.3.255 scope global vlan30
17: vlan40: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.4.6/24 brd 172.24.4.255 scope global vlan40
18: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.5.6/24 brd 172.24.5.255 scope global vlan50
```

4.2. Use the **`ovs-vsctl`** command to list the bridges on the `computehci0` node.

```
[heat-admin@computehci0 ~]$ sudo ovs-vsctl list-br
br-eth3
br-eth4
br-ex
br-int
br-trunk
```

4.3. Use the **`ovs-vsctl`** command to list the interfaces on `br-trunk`.

```
[heat-admin@computehci0 ~]$ sudo ovs-vsctl list-ifaces br-trunk
eth1
vlan10
vlan20
vlan30
vlan40
vlan50
```

4.4. Use the **`docker`** command to list the containers on the `computehci0` node.

```
[heat-admin@computehci0 ~]$ sudo docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                       STATUS
ceph-osd-computehci0-vdc    Up 7 hours
ceph-osd-computehci0-vdd    Up 7 hours
ceph-osd-computehci0-vdb    Up 7 hours
ovn_controller              Up 7 hours
ovn_metadata_agent          Up 7 hours (healthy)
logrotate_crond             Up 7 hours
nova_compute                Up 7 hours (healthy)
nova_migration_target       Up 7 hours
ceilometer_agent_compute    Up 7 hours
iscsid                      Up 7 hours
nova_libvirt                Up 7 hours
nova_virtlogd               Up 7 hours
```

4.5. Use the **`lsblk -fs`** command to list the specified block devices. Log off from the `computehci0` node.

```
[heat-admin@computehci0 ~]$ lsblk -fs
NAME  FSTYPE  LABEL      UUID                                   MOUNTPOINT
sr0   iso9660 iPXE       2014-12-21-12-02-07-00
vda1  iso9660 config-2   2018-08-19-03-13-46-00
└─vda
vda2  xfs     img-rootfs afa067cb-d91f-432e-b7c4-c546dabe10ec /
└─vda
vdb1  xfs                77e922d3-e888-4a9c-9cc4-eb0f9025dcbc
└─vdb
```

```
vdb2
└─vdb
vdc1  xfs                 3f8498e3-4c0b-4495-ab8c-0cfbdd099549
└─vdc
vdc2
└─vdc
vdd1  xfs                 9c995a6f-32c5-4138-90da-dd4239ce37ec
└─vdd
vdd2
└─vdd
[heat-admin@computehci0 ~]$ exit
(undercloud) [stack@director ~]$
```

5. Log in to the `ceph0` node as the `heat-admin` user. List the relevant services and network configuration, and then log off.

5.1. Log in to the `ceph0` node as the `heat-admin` user. Use the **`ip addr`** command to list the network configuration.

```
(undercloud) [stack@director ~]$ ssh heat-admin@ceph0
[heat-admin@ceph0 ~]$ ip addr | grep -E 'eth0|vlan|eth2'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
    inet 172.25.249.56/24 brd 172.25.249.255 scope global eth0
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
6: vlan40: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.4.3/24 brd 172.24.4.255 scope global vlan40
7: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.3.3/24 brd 172.24.3.255 scope global vlan30
8: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.5.3/24 brd 172.24.5.255 scope global vlan50
```

5.2. Use the **`ovs-vsctl`** command to list the bridges on the `ceph0` node.

```
[heat-admin@ceph0 ~]$ sudo ovs-vsctl list-br
br-ex
br-trunk
```

5.3. Use the **`docker`** command to list the containers on the `ceph0` node. Log off from the `ceph0` node.

```
[heat-admin@ceph0 ~]$ sudo docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                STATUS
ceph-osd-ceph0-vdb   Up 7 hours
ceph-osd-ceph0-vdc   Up 7 hours
ceph-osd-ceph0-vdd   Up 7 hours
logrotate_crond      Up 7 hours
```

5.4. Use the **`lsblk -fs`** command to list the specified block devices.

```
[heat-admin@ceph0 ~]$ lsblk -fs
NAME  FSTYPE  LABEL      UUID                                 MOUNTPOINT
sr0   iso9660 iPXE       2014-12-21-12-02-07-00
vda1  iso9660 config-2   2018-08-19-03-13-46-00
└─vda
```

```
vda2  xfs     img-rootfs afa067cb-d91f-432e-b7c4-c546dabe10ec /
└─vda
vdb1  xfs                c3d437d0-9215-4c98-8a40-c5dacabf95b1
└─vdb
vdb2
└─vdb
vdc1  xfs                d76e086b-db1e-4a84-9b7e-e23a51f71eee
└─vdc
vdc2
└─vdc
vdd1  xfs                1c4c6a2b-0b99-4453-8821-caea37e78728
└─vdd
vdd2
└─vdd
[heat-admin@ceph0 ~]$ exit
(undercloud) [stack@director ~]$
```

6. Log in to the `controller0` node. Use Ceph commands to check the health of the Ceph cluster.

   6.1. Log in to the `controller0` node. Use the **`ceph status`** command to check the health of the Ceph cluster.

```
(undercloud) [stack@director ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ ceph status
  cluster:
    id:     f9d61d02-a355-11e8-ae2b-52540001fac8
    health: HEALTH_OK

  services:
    mon: 1 daemons, quorum controller0
    mgr: controller0(active)
    mds: cephfs-1/1/1 up  {0=controller0=up:active}
    osd: 6 osds: 6 up, 6 in

  data:
    pools:   7 pools, 416 pgs
    objects: 744 objects, 5617 MB
    usage:   6270 MB used, 107 GB / 113 GB avail
    pgs:     416 active+clean
```

   6.2. Use the **`ceph osd lspools`** command to list the cluster's pools.

```
[heat-admin@controller0 ~]$ ceph osd lspools
1 images,2 metrics,3 backups,4 vms,5 volumes,6 manila_data,7 manila_metadata,
```

   6.3. Use the **`ceph osd ls`** command to list the cluster's OSDs. Log off from the `ceph0` node.

```
[heat-admin@controller0 ~]$ ceph osd ls
0
1
2
3
4
```

```
5
[heat-admin@controller0 ~]$ exit
(undercloud) [stack@director ~]$
```

7. On the `director` node, authenticate as the `admin` user in the **`admin`** project in the overcloud.

   Source the **`overcloudrc`** authentication environment file. The loaded environment provides `admin` user access in the overcloud.

```
(undercloud) [stack@director ~]$ source overcloudrc
```

8. Confirm that the `heat_stack_user` role is available in the overcloud. Log out of the `director` node.

```
(overcloud) [stack@director ~]$ openstack role list -c Name
+-----------------+
| Name            |
+-----------------+
| heat_stack_user |
| ResellerAdmin   |
| _member_        |
| swiftoperator   |
| admin           |
+-----------------+
```

```
(overcloud) [stack@director ~]$ exit
```

Cleanup

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

```
[student@workstation ~]$ lab architecture-overcloud cleanup
```

This concludes the guided exercise.

NAVIGATING THE RED HAT OPENSTACK PLATFORM ARCHITECTURE

PERFORMANCE CHECKLIST

In this lab, you will validate that the undercloud and overcloud are functional by navigating the classroom infrastructure, listing networks and interfaces, and finally by deploying a server instance with a floating IP address.

OUTCOMES

You should be able to:

- Navigate the classroom infrastructure.
- List the networks and interfaces on the overcloud nodes.
- Manage the containers using basic commands.
- List the OVS bridges on the overcloud nodes.
- Deploy and verify an external instance.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab architecture-lab setup`** command. The script checks that the `default` flavor, the `rhel7` image, and the `provider-172.25.250` network exist to test instance deployment.

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

1. On the `director` node, verify the networks and services of the undercloud.
2. On the `director` node, find the IP addresses of the overcloud nodes. List the networks and show the details for the `ctlplane` subnet. Open the **`undercloud.conf`** file and look at the DHCP configuration.
3. On all four overcloud nodes, use the **`docker`** command to display the OpenStack services.
4. Use **`docker`** commands to stop the `nova_api` container. Review the status of the container and then restart it. Ensure that the container returns a **`healthy`** status.
5. On the `controller0` and `compute0` nodes, view the `eth0`, `vlan`, and `br-ex` interfaces. List the OVS bridges. List the OVS interfaces on `br-ex` and `br-trunk`.
6. On `workstation`, source the **`~/operator1-production-rc`** keystone file. Create an instance in the **`production`** project using the attributes in the table below. Ensure that the server has a status of **`ACTIVE`**. Create a floating IP address on the `provider-datacentre` network and attach it to the instance.

Instance Attributes

ATTRIBUTE	VALUE
Flavor	default
Key pair	example-keypair
Network	production-network1
Image	production-rhel7
Security group	production
Name	production-server1
User data	http://materials.example.com/oscli-setup.sh

Evaluation

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

```
[student@workstation ~(operator1-production)]$ lab architecture-lab grade
```

Cleanup

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

```
[student@workstation ~(operator1-production)]$ lab architecture-lab cleanup
```

This concludes the lab.

NAVIGATING THE RED HAT OPENSTACK PLATFORM ARCHITECTURE

PERFORMANCE CHECKLIST

In this lab, you will validate that the undercloud and overcloud are functional by navigating the classroom infrastructure, listing networks and interfaces, and finally by deploying a server instance with a floating IP address.

OUTCOMES

You should be able to:

- Navigate the classroom infrastructure.
- List the networks and interfaces on the overcloud nodes.
- Manage the containers using basic commands.
- List the OVS bridges on the overcloud nodes.
- Deploy and verify an external instance.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab architecture-lab setup`** command. The script checks that the `default` flavor, the `rhel7` image, and the `provider-172.25.250` network exist to test instance deployment.

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

1. On the `director` node, verify the networks and services of the undercloud.

   1.1. Log in to `director` as the `stack` user. Use the **`openstack network list`** command to list the networks built on the undercloud. Note the external network, the provisioning network (ctlplane), and the five VLAN networks.

```
[student@workstation ~]$ ssh stack@director
(undercloud) [stack@director ~]$ openstack network list
+-------------+--------------+-------------+
| ID          | Name         | Subnets     |
+-------------+--------------+-------------+
| 057a...250d | tenant       | e2fc...f6c0 |
| 0cf6...e3d0 | management   | 5800...11e5 |
| 3a82...e1f5 | ctlplane     | a8e0...9913 |
| 3e25...7d07 | storage_mgmt | 1f31...d167 |
| 6625...62e7 | storage      | e312...7347 |
| 6d06...159d | external     | 3720...90d6 |
| a432...0a0e | internal_api | 7c19...4549 |
```

```
+-------------+-------------+-------------+
```

1.2. Use the **`ip addr`** command to list the interfaces on `director`. Note the `eth1` interface, which links to the external network. The `br-ctlplane` interface links to the provisioning network.

```
(undercloud) [stack@director ~]$ ip a | grep -E 'br-ctlplane|eth1'
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
  inet 172.25.250.200/24 brd 172.25.250.255 scope global noprefixroute eth1
6: br-ctlplane: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UNKNOWN group default qlen 1000
  inet 172.25.249.200/24 brd 172.25.249.255 scope global br-ctlplane
  inet 172.25.249.202/32 scope global br-ctlplane
  inet 172.25.249.201/32 scope global br-ctlplane
```

1.3. Use the **`openstack compute service list`** command to list the `nova` services. All of the services must be enabled and up for the environment to work correctly.

```
(undercloud) [stack@director ~]$ openstack compute service list \
-c Binary -c Host -c Status -c State
+----------------+--------------------------+---------+-------+
| Binary         | Host                     | Status  | State |
+----------------+--------------------------+---------+-------+
| nova-conductor | director.lab.example.com | enabled | up    |
| nova-scheduler | director.lab.example.com | enabled | up    |
| nova-compute   | director.lab.example.com | enabled | up    |
+----------------+--------------------------+---------+-------+
```

1.4. Use the **`openstack catalog list`** command to list all of the `ironic` endpoints. Note that each service has three different endpoints.

```
(undercloud) [stack@director ~]$ openstack catalog list -c Name -c Endpoints
+------------------+----------------------------------------+
| Name             | Endpoints                              |
+------------------+----------------------------------------+
| ironic-inspector | regionOne                              |
|                  |   public: https://172.25.249.201:13050 |
|                  | regionOne                              |
|                  |   internal: http://172.25.249.202:5050 |
|                  | regionOne                              |
|                  |   admin: http://172.25.249.202:5050    |
|                  |                                        |
| mistral          | regionOne                              |
|                  |   internal: http://172.25.249.202:8989/v2|
|                  | regionOne                              |
|                  |   admin: http://172.25.249.202:8989/v2 |
|                  | regionOne                              |
|                  |   public: https://172.25.249.201:13989/v2|
...output omitted...
```

2. On the `director` node, find the IP addresses of the overcloud nodes. List the networks and show the details for the `ctlplane` subnet. Open the **`undercloud.conf`** file and look at the DHCP configuration.

   2.1. Use the **`openstack server list`** command to list the overcloud nodes' IP addresses.

```
(undercloud) [stack@director ~]$ openstack server list -c Name -c Networks
+-------------+------------------------+
| Name        | Networks               |
+-------------+------------------------+
| compute1    | ctlplane=172.25.249.57 |
| ceph0       | ctlplane=172.25.249.56 |
| controller0 | ctlplane=172.25.249.53 |
| compute0    | ctlplane=172.25.249.54 |
| computehci0 | ctlplane=172.25.249.58 |
+-------------+------------------------+
```

   2.2. Use the **`openstack network list`** command to list the undercloud networks. Use the **`openstack subnet show`** command to show the details of the `ctlplane` subnet. Note the content of the **`allocation_pools`** parameter.

```
(undercloud) [stack@director ~]$ openstack network list -c Name -c Subnets
+--------------+--------------------------------------+
| Name         | Subnets                              |
+--------------+--------------------------------------+
| tenant       | ca617bee-fbb4-46be-868a-97b6384c9202 |
| external     | d49717b4-1327-4359-852b-c4e600ceba97 |
| internal_api | e786ca9b-5390-4197-b4f7-d5bdb8efa309 |
| storage_mgmt | a2706edf-0080-44f4-b41a-ac7198a920a0 |
| management   | 746b8a96-3c66-4421-8cbb-30935754f82a |
| ctlplane     | 6457fb0f-2d9d-43af-93ca-f0c3dc008c82 |
| storage      | 0b695964-76c5-48d0-a6c0-6ac6c9d6465c |
+--------------+--------------------------------------+
(undercloud) [stack@director ~]$ openstack subnet show \
6457fb0f-2d9d-43af-93ca-f0c3dc008c82
+-------------------+--------------------------------+
| Field             | Value                          |
+-------------------+--------------------------------+
| allocation_pools  | 172.25.249.51-172.25.249.59    |
...output omitted...
```

   2.3. Search for the **`dhcp_start`** and **`dhcp_end`** parameter IP adresses in the **`undercloud.conf`** file.

```
(undercloud) [stack@director ~]$ grep '^dhcp_' undercloud.conf
dhcp_start = 172.25.249.51
dhcp_end = 172.25.249.59
```

   Note that all overcloud node IP addresses are within the allocation pool on the `ctlplane` subnet. The `ctlplane` subnet is configured in the **`undercloud.conf`** file. Overcloud nodes IP addresses are obtained from the **`allocation_pools`** option of the `ctlplane` subnet.

3. On all four overcloud nodes, use the **docker** command to display the OpenStack services.

   3.1. Log in to the `controller0` node as the `heat-admin` user. Gain `root` privileges using the **sudo -i** command. Use the **docker ps** command to list the containers on the controller node. Exit the `controller0` node.

```
(undercloud) [stack@director ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                              STATUS
ovn-dbs-bundle-docker-0            Up 7 hours
openstack-manila-share-docker-0    Up 7 hours
openstack-cinder-volume-docker-0   Up 7 hours
haproxy-bundle-docker-0            Up 7 hours
redis-bundle-docker-0              Up 7 hours
galera-bundle-docker-0             Up 7 hours
...output omitted...
[root@controller0 ~]# exit
[heat-admin@controller0 ~]$ exit
```

   3.2. Log in to the `compute0` node. Use the **sudo -i** command to gain `root` privileges. On the `compute0` node, run the **docker ps** command to list the containers. Exit the `compute0` node and log in to the `computehci0` node. Gain `root` privileges using the **sudo -i** command and run the **docker ps** command. Note the difference between the two nodes. Exit the `computehci0` node.

```
(undercloud) [stack@director ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]# docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                      STATUS
ovn_metadata_agent         Up 8 hours (healthy)
ovn_controller             Up 8 hours
nova_compute               Up 8 hours (healthy)
logrotate_crond            Up 8 hours
ceilometer_agent_compute   Up 8 hours
nova_migration_target      Up 8 hours
iscsid                     Up 8 hours
nova_libvirt               Up 8 hours
nova_virtlogd              Up 8 hours
[root@compute0 ~]# exit
[heat-admin@compute0 ~]$ exit
(undercloud) [stack@director ~]$ ssh heat-admin@computehci0
[heat-admin@computehci0 ~]$ sudo -i
[root@computehci0 ~]# docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                      STATUS
ceph-osd-computehci0-vdc   Up 8 hours
ceph-osd-computehci0-vdd   Up 8 hours
ceph-osd-computehci0-vdb   Up 8 hours
ovn_controller             Up 8 hours
ovn_metadata_agent         Up 8 hours (healthy)
logrotate_crond            Up 8 hours
nova_compute               Up 8 hours (healthy)
nova_migration_target      Up 8 hours
```

```
ceilometer_agent_compute   Up 8 hours
iscsid                     Up 8 hours
nova_libvirt               Up 8 hours
nova_virtlogd              Up 8 hours
[root@computehci0 ~]# exit
[heat-admin@computehci0 ~]$ exit
```

3.3. Log in to `ceph0` as the `heat-admin` user. Gain `root` privileges using the **`sudo -i`** command. Use the **`docker ps`** command to list the containers on the `ceph0` node. Exit the `ceph0` node.

```
(undercloud) [stack@director ~]$ ssh heat-admin@ceph0
[heat-admin@ceph0 ~]$ sudo -i
[root@ceph0 ~]# docker ps \
--format="table {{.Names}}\t{{.Status}}"
NAMES                STATUS
ceph-osd-ceph0-vdb   Up 8 hours
ceph-osd-ceph0-vdc   Up 8 hours
ceph-osd-ceph0-vdd   Up 8 hours
logrotate_crond      Up 8 hours
[root@ceph0 ~]# exit
[heat-admin@ceph0 ~]$ exit
logout
Connection to ceph0 closed.
(undercloud) [stack@director ~]$
```

4. Use **`docker`** commands to stop the `nova_api` container. Review the status of the container and then restart it. Ensure that the container returns a **`healthy`** status.

4.1. Log in to `controller0` as the `heat-admin` user. Use the **`docker stop`** command to stop the `nova_api` container. Verify the container status using **`docker ps`**. Use the **`docker start`** command to restart the `nova_api` container. Recheck the container status, waiting as needed, until the container status is **`healthy`**.

```
[student@workstation ~]$ ssh heat-admin@controller0
Last login: Mon Aug 27 12:14:20 2018 from 172.25.250.200
[heat-admin@controller0 ~]$
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# docker stop nova_api
nova_api
[root@controller0 ~]# docker ps -a \
--format="table {{.Names}}\t{{.Status}}" | grep nova_api
...output omitted...
nova_api                          Exited (0) 2 minutes ago
...output omitted...
[root@controller0 ~]# docker start nova_api
nova_api
[root@controller0 ~]# docker ps \
--format="table {{.Names}}\t{{.Status}}" | grep nova_api
nova_api                          Up 26 seconds (health: starting)
...output omitted...
[root@controller0 ~]# docker ps \
--format="table {{.Names}}\t{{.Status}}" | grep nova_api
nova_api                          Up 31 seconds (healthy)
...output omitted...
```

5. On the `controller0` and `compute0` nodes, view the `eth0`, `vlan`, and `br-ex` interfaces. List the OVS bridges. List the OVS interfaces on `br-ex` and `br-trunk`.

   5.1. On the `controller0` node, list the `eth0`, `vlan`, and `br-ex` interfaces. List the OVS bridges and the interfaces on `br-ex` and `br-trunk`. Exit the `controller0` system.

```
(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 172.25.249.59/24 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
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: 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
16: 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
17: 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
18: 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-br
br-eth3
br-eth4
br-ex
br-int
br-trunk
[heat-admin@controller0 ~]$ sudo ovs-vsctl list-ifaces br-trunk
eth1
vlan10
vlan20
vlan30
vlan40
vlan50
[heat-admin@controller0 ~]$ sudo ovs-vsctl list-ifaces br-ex
eth2
patch-provnet-f8366a5d-9cf9-4d38-9760-9cf9df4eceaa-to-br-int
[heat-admin@controller0 ~]$ exit
```

   5.2. Log in to `compute0` as the `heat-admin` user. List the `eth0`, `vlan`, and `br-ex` interfaces. List the OVS bridges and the interfaces on `br-ex` and `br-trunk`. Exit the `compute0` system.

```
(undercloud) [stack@director ~]$ ssh heat-admin@compute0
 [heat-admin@compute0 ~]$ ip addr | grep -E 'eth0|vlan|eth2'
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state
    inet 172.25.249.53/24 brd 172.25.249.255 scope global eth0
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state
```

```
14: vlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.1.2/24 brd 172.24.1.255 scope global vlan10
15: vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.2.2/24 brd 172.24.2.255 scope global vlan20
16: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.3.2/24 brd 172.24.3.255 scope global vlan30
17: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
    inet 172.24.5.2/24 brd 172.24.5.255 scope global vlan50
[heat-admin@compute0 ~]$ sudo ovs-vsctl list-br
br-eth3
br-eth4
br-ex
br-int
br-trunk
[heat-admin@compute0 ~]$ sudo ovs-vsctl list-ifaces br-trunk
eth1
vlan10
vlan20
vlan30
vlan50
[heat-admin@compute0 ~]$ exit
```

6. On `workstation`, source the **`~/operator1-production-rc`** keystone file. Create an instance in the **`production`** project using the attributes in the table below. Ensure that the server has a status of **`ACTIVE`**. Create a floating IP address on the `provider-datacentre` network and attach it to the instance.

Instance Attributes

ATTRIBUTE	VALUE
Flavor	default
Key pair	example-keypair
Network	production-network1
Image	production-rhel7
Security group	production
Name	production-server1
User data	http://materials.example.com/oscli-setup.sh

6.1. Source the **`~/operator1-production-rc`** file and launch an instance as the `operator1` user. Name the instance `production-server1`.

```
[student@workstation ~]$ source operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=production-network1 \
--security-group default \
--image production-rhel7 \
production-server1 --wait
```

```
+-----------------------------+--------------------------------+
| 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      | 2018-08-27T13:39:06.000000     |
| OS-SRV-USG:terminated_at    | None                           |
| accessIPv4                  |                                |
| accessIPv6                  |                                |
| addresses                   | production-network1=192.168.1.1 |
| adminPass                   | eCkjYF6hkH9Q                   |
| config_drive                |                                |
| created                     | 2018-08-27T13:38:16Z           |
| flavor                      | default (c620...6c0f)          |
| hostId                      | 8bb2...d4e4                    |
| id                          | c82c...15ee                    |
| image                       | production-rhel7 (d4c2...7be0)  |
| key_name                    | example-keypair                |
| name                        | production-server1             |
| progress                    | 0                              |
| project_id                  | ee1b8ae964ee4ab3b2edf3fbc7791a40|
| properties                  |                                |
| security_groups             | name='default'                 |
| status                      | ACTIVE                         |
| updated                     | 2018-08-27T13:39:06Z           |
| user_id                     | 9951c5c09af6486c8fa6df513be889f1|
| volumes_attached            |                                |
+-----------------------------+--------------------------------+
[student@workstation ~(operator1-production)]$ openstack server list \
-c Name -c Status
+--------------------+--------+
| Name               | Status |
+--------------------+--------+
| production-server1 | ACTIVE |
+--------------------+--------+
```

6.2. Use the **`openstack floating ip create`** command to create a floating IP address in the `provider-datacentre` network.

```
[student@workstation ~(operator1-production)]$ openstack floating ip create \
provider-datacentre
+---------------------+-----------------------+
| Field               | Value                 |
+---------------------+-----------------------+
| created_at          | 2018-08-27T13:20:58Z  |
| description         |                       |
| fixed_ip_address    | None                  |
| floating_ip_address | 172.25.250.N          |
| floating_network_id | f836...ceaa           |
| id                  | 9779...4ed8           |
| name                | 172.25.250.113        |
| port_id             | None                  |
| project_id          | ee1b...1a40           |
```

```
| qos_policy_id        | None                   |
| revision_number      | 0                      |
| router_id            | None                   |
| status               | DOWN                   |
| subnet_id            | None                   |
| updated_at           | 2018-08-27T13:20:58Z   |
+----------------------+------------------------+
```

6.3. Attach the floating IP address to the active server and verify that it responds.

```
[student@workstation ~(operator1-production)]$ openstack server add \
floating ip production-server1 172.25.250.N
[student@workstation ~(operator1-production)]$ ping -c3 172.25.250.N
PING 172.25.250.113 (172.25.250.113) 56(84) bytes of data.
64 bytes from 172.25.250.113: icmp_seq=1 ttl=63 time=0.346 ms
64 bytes from 172.25.250.113: icmp_seq=2 ttl=63 time=0.403 ms
64 bytes from 172.25.250.113: icmp_seq=3 ttl=63 time=0.375 ms

--- 172.25.250.113 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2000ms
rtt min/avg/max/mdev = 0.346/0.374/0.403/0.032 ms
```

6.4. Log into the instance using the attached address. Test for external network access. Ping the network gateway from `production-server1`.

```
[student@workstation ~(operator1-production)]$ ssh cloud-user@172.25.250.N
[cloud-user@production-server1 ~]$ ping -c3 172.25.250.254
PING 172.25.250.254 (172.25.250.254) 56(84) bytes of data.
64 bytes from 172.25.250.254: icmp_seq=1 ttl=63 time=0.804 ms
64 bytes from 172.25.250.254: icmp_seq=2 ttl=63 time=0.847 ms
64 bytes from 172.25.250.254: icmp_seq=3 ttl=63 time=0.862 ms

--- 172.25.250.254 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.804/0.837/0.862/0.041 ms
```

6.5. When finished testing, exit the `production-server1` server instance.

```
[cloud-user@production-server1 ~]$ exit
[student@workstation ~(operator1-production)]$
```

Evaluation

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

```
[student@workstation ~(operator1-production)]$ lab architecture-lab grade
```

Cleanup

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

```
[student@workstation ~(operator1-production)]$ lab architecture-lab cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Enterprise clouds today are built using multiple, interconnected cloud structures. The undercloud is a provisioning and management cloud for building and managing the production clouds. Red Hat OpenStack Platform Director is the undercloud in Red Hat OpenStack Platform.
- An enterprise production cloud is known as an overcloud. Underclouds and overclouds utilize the same technologies, but manage different workloads. Underclouds manage cloud infrastructure, and overclouds manage production and tenant workloads.
- Common open technologies are used in physical and virtual clouds. Intelligent Platform Management Interface (IPMI) is the power management technology used to control nodes. Virtual Network Computing (VNC) is the remote access technology used to access deployed instance consoles.
- The introspection process discovers the technical characteristics of nodes to be deployed. Using those characteristics, overcloud deployment can automatically assign deployment roles to specific nodes.
- The orchestration process defines the specific configuration for each node's hardware and software. The provided default templates cover a majority of common use cases and designs.
- Recent versions of Red Hat OpenStack Platform use containers to run services. The **`systemd`** commands are deprecated and have been replaced with **`docker`** commands. Log files are now located in **`/var/log/`*`service`***. Configuration files are now located in **`/var/lib/config-data/puppet-generated/`*`service`*`/etc/`*`service`***. Only changes made in the container configuration files are persistent over a restart of the service.

CHAPTER 2

DESCRIBING THE OPENSTACK CONTROL PLANE

GOAL

Introduce the critical core services making up the OpenStack control plane.

OBJECTIVES

- Describe service endpoint configuration and security.
- Describe message broker patterns and implementation.
- Identify the shared services running on a controller node.

SECTIONS

- Describing Service Access by API (and Quiz)
- Describing Service Communication by Message Broker (and Guided Exercise)
- Identifying Overcloud Control Plane Services (and Guided Exercise)

LAB

- Describing the OpenStack Control Plane

DESCRIBING SERVICE ACCESS BY API

OBJECTIVES

After completing this section, students should be able to describe service endpoint configuration and security.

KEYSTONE SERVICE CATALOG

The service catalog is a crucial element of the Keystone architecture. It provides a list of endpoint URLs that can be dynamically discovered by API clients, and provides the service endpoints that can be accessed by a token. Without a service catalog, API clients would be unaware of which URL an API request should use. The **`openstack catalog show`** command displays catalog information for a service. The service name is passed as an argument to the command. For example, to view the service catalog data for Nova compute, use the following command:

```
[user@demo ~(admin)]$ openstack catalog show nova
+-----------+-----------------------------------------------------+
| Field     | Value                                               |
+-----------+-----------------------------------------------------+
| endpoints | regionOne (1)                                       |
|           |        (2) publicURL: http://172.25.250.50:8774/v2.1 |
|           |          internalURL: http://172.24.1.50:8774/v2.1 |
|           |          adminURL: http://172.24.1.50:8774/v2.1    |
|           |                                                     |
| name      | nova (3)                                            |
| type      | compute (4)                                         |
+-----------+-----------------------------------------------------+
```

1. The region for the URLs.
2. The list of internal, public, and admin URLs by which an API client request can access the Nova compute service.
3. The user-facing service name.
4. The OpenStack registered type, such as image-service and object-store.

ENDPOINTS

An endpoint is a URL that an API client uses to access a service in OpenStack. Every service has one or more endpoints. There are three types of endpoint URLs: admin, public, and internal. The admin URL should only be used by those who require administrative access to a service endpoint. The internal URL is used by services to communicate with each other on a network that is unmetered or free of bandwidth charges. The public URL is designed with the intention of being consumed by end users from a public network.

To list all services and their endpoints, use the **`openstack catalog list`** command as the `admin` user.

```
[user@demo ~(admin)]$ openstack catalog list
+---------+---------+-----------------------------------------------+
| Name    | Type    | Endpoints                                     |
```

```
+---------+---------+----------------------------------------------+
| nova    | compute | regionOne                                    |
|         |         |   publicURL: https://172.25.249.201:13774/v2.1 |
|         |         |   internalURL: http://172.25.249.200:8774/v2.1 |
|         |         |   adminURL: http://172.25.249.200:8774/v2.1    |
|         |         |                                              |
| neutron | network | regionOne                                    |
|         |         |   publicURL: https://172.25.249.201:13696      |
|         |         |   internalURL: http://172.25.249.200:9696      |
|         |         |   adminURL: http://172.25.249.200:9696         |
...output omitted...
```

To list the ID, region, service name, and service type of all endpoints, use the **`openstack endpoint list`** command.

```
[user@demo ~(admin)]$ openstack endpoint list
+----------------------------------+-----------+--------------+----------------+
| ID                               | Region    | Service Name | Service Type   |
+----------------------------------+-----------+--------------+----------------+
| d1812da138514794b27d266a22f66b15 | regionOne | aodh         | alarming       |
| b1484c933ba74028965a51d4d0aa9f04 | regionOne | nova         | compute        |
| 4c6117b491c243aabbf40d7dfdf5ce9a | regionOne | heat-cfn     | cloudformation |
| eeaa5964c26042e38c632d1a12e001f3 | regionOne | heat         | orchestration  |
| 1aeed510fa9a433795a4ab5db80e19ec | regionOne | glance       | image          |
...output omitted...
```

TROUBLESHOOTING API COMMUNICATION

A correct catalog and endpoint configuration is essential for the OpenStack environment to function effectively. Common issues that lead to troubleshooting are misconfigured endpoints and user authentication. There is a known issue documented in BZ-1404324 where the scheduled token flushing job is not effective enough for large deployments, we will review the fix in the following guided exercise. When issues do arise, you can take steps to investigate and find a resolution to the issue. The following is a list of troubleshooting steps:

- Ensure the authentication credentials and token are appropriate using the **`curl`** command to retrieve the service catalog.

```
[user@demo ~(admin)]$  curl -s -X POST http://172.25.250.50:5000/v3/token \
-d '{"auth": {"passwordCredentials": {"username":"admin", \
"password":"Y7Q72DfAjKjUgA2G87yHEJ2Bz"}, "tenantName":"admin"}}' \
-H "Content-type: application/json" | jq .
{
  "access": {
    "metadata": {
      "roles": [
        "f79b6d8bfada4ab89a7d84ce4a0747ff"
      ],
      "is_admin": 0
    },
    "user": {
      "name": "admin",
      "roles": [
        {
          "name": "admin"
```

```
        }
      ],
      "id": "15ceac73d7bb4437a34ee26670571612",
      "roles_links": [],
      "username": "admin"
    },
    "serviceCatalog": [
      {
        "name": "nova",
        "type": "compute",
        "endpoints_links": [],
        "endpoints": [
...output omitted...
```

- Inspect the **/var/log/containers/keystone/keystone.log** file for **[Errno 111] Connection refused** errors. This indicates there is an issue connecting to a service endpoint.

```
2017-06-04 14:07:49.332 2855 ERROR oslo.messaging._drivers.impl_rabbit
[req-1b8d5196-d787-49db-be60-025ce0ab575d - - - - -] [73809126-9833-487a-
a69a-4a7d9dffd08c]
AMQP server on 172.25.249.200:5672 is unreachable: [Errno 111] Connection refused.
Trying again in 1 seconds. Client port: None
```

- Every service has an API log that should be inspected when troubleshooting endpoints. For example, if an operator cannot retrieve Glance image data, an inspection of **/var/log/containers/glance/api.log** may provide useful information. Query the file for **DiscoveryFailure**.

```
DiscoveryFailure: Could not determine a suitable URL for the plugin
2017-05-30 04:31:17.650 277258 INFO eventlet.wsgi.server [-] 172.24.3.1 - - [30/
May/2017 04:31:17] "GET /v2/images HTTP/1.1" 500 139 0.003257
```

- Include the **--debug** option to the **openstack catalog show** command (or to any **openstack** command) to view the HTTP request from the client and the responses from the endpoints. This example shows the request and response for the Identity endpoint, followed by the request and response for a token, then the command to obtain the endpoint information. The JSON payloads have been formatted here for readability.

```
[user@demo ~(admin)]$ openstack catalog show nova --debug
...output omitted...
Get auth_ref
REQ: curl -g -i -X GET http://172.25.250.50:5000//v3 -H "Accept: application/
json" -H "User-Agent: osc-lib/1.9.0 keystoneauth1/3.4.0 python-requests/2.14.2
 CPython/2.7.5"
Starting new HTTP connection (1): 172.25.250.50
http://172.25.250.50:5000 "GET //v3 HTTP/1.1" 200 195
RESP: [200] Date: Tue, 21 Aug 2018 01:34:07 GMT Server: Apache Vary: X-Auth-
Token,Accept-Encoding x-openstack-request-id: req-40fbcc65-6b95-4538-bccf-
d2a316d198cc Content-Encoding: gzip Content-Length: 195 Content-Type: application/
json
RESP BODY:
{
    "version": {
        "id": "v3.10",
        "links": [
```

```
            {
                "href": "http://172.25.250.50:5000/v3/",
                "rel": "self"
            }
        ],
        "media-types": [
            {
                "base": "application/json",
                "type": "application/vnd.openstack.identity-v3+json"
            }
        ],
        "status": "stable",
        "updated": "2018-02-28T00:00:00Z"
    }
}

GET call to None for http://172.25.250.50:5000//v3 used request id
 req-40fbcc65-6b95-4538-bccf-d2a316d198cc
Making authentication request to http://172.25.250.50:5000/v3/auth/tokens
http://172.25.250.50:5000 "POST /v3/auth/tokens HTTP/1.1" 201 10790
{
    "token": {
        "audit_ids": [
            "-paEQLiQRx2xxIAwyGtWWg"
        ],
        "catalog": [
            {
                "endpoints": [
                    {
                        "id": "414c1a2d92d344e59044caa4c2ad4aed",
                        "interface": "admin",
                        "region": "regionOne",
                        "region_id": "regionOne",
                        "url": "http://172.24.1.50:9876"
                    },
                    {
                        "id": "8e370f1219214534b574576823c576f3",
                        "interface": "public",
                        "region": "regionOne",
                        "region_id": "regionOne",
                        "url": "http://172.25.250.50:9876"
                    },
...output omitted...
        "expires_at": "2018-08-21T17:34:08.000000Z",
        "is_domain": false,
        "issued_at": "2018-08-21T01:34:08.000000Z",
        "methods": [
            "password"
        ],
        "project": {
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "9a77004b07374d6bb32761116e1e489a",
            "name": "admin"
```

```
        },
        "roles": [
            {
                "id": "a46d00e8f7cb4e2295ffcd832e366c82",
                "name": "admin"
            }
        ],
        "user": {
            "domain": {
                "id": "default",
                "name": "Default"
            },
            "id": "f8c4979f438b47339574e5cfdbe336d3",
            "name": "admin",
            "password_expires_at": null
        }
    }
}

run(Namespace(columns=[], fit_width=False, formatter='table', max_width=0,
 noindent=False, prefix='', print_empty=False, service=u'nova', variables=[]))
+-----------+-----------------------------------------+
| Field     | Value                                   |
+-----------+-----------------------------------------+
| endpoints | regionOne                               |
|           |   public: http://172.25.250.50:8774/v2.1 |
|           | regionOne                               |
|           |   internal: http://172.24.1.50:8774/v2.1 |
|           | regionOne                               |
|           |   admin: http://172.24.1.50:8774/v2.1    |
|           |                                         |
| id        | 32d59110934045848e7598578aa58310        |
| name      | nova                                    |
| type      | compute                                 |
+-----------+-----------------------------------------+
clean_up ShowCatalog:
END return value: 0
```

REFERENCES

Identity Concepts
https://docs.openstack.org/keystone/latest/admin/identity-concepts.html

API endpoint configuration
https://docs.openstack.org/security-guide/api-endpoints/api-endpoint-configuration-recommendations.html

DESCRIBING SERVICE ACCESS BY API

Choose the correct answers to the following questions:

When you have completed the quiz, click CHECK. If you wish to try again, click RESET. Click SHOW SOLUTION to see all of the correct answers.

1. Which option to the `openstack` command displays HTTP requests and responses?

a. `--trace`
b. `--verbose`
c. `--debug`
d. `--headers`

2. What is the purpose of the Keystone service catalog?

a. To allow clients to authenticate from a remote location
b. To allow clients to place orders for custom virtual machines
c. To allow clients to discover the available service endpoints
d. To allow clients to allocate more compute resources dynamically

3. Which three of the following are valid endpoint types? (Choose three.)

a. externalURL
b. internalURL
c. adminURL
d. masterURL
e. publicURL

DESCRIBING SERVICE ACCESS BY API

Choose the correct answers to the following questions:

When you have completed the quiz, click CHECK. If you wish to try again, click RESET. Click SHOW SOLUTION to see all of the correct answers.

1. Which option to the `openstack` command displays HTTP requests and responses?

a. `--trace`
b. `--verbose`
c. `--debug`
d. `--headers`

2. What is the purpose of the Keystone service catalog?

a. To allow clients to authenticate from a remote location
b. To allow clients to place orders for custom virtual machines
c. To allow clients to discover the available service endpoints
d. To allow clients to allocate more compute resources dynamically

3. Which three of the following are valid endpoint types? (Choose three.)

a. externalURL
b. internalURL
c. adminURL
d. masterURL
e. publicURL

DESCRIBING SERVICE COMMUNICATION BY MESSAGE BROKER

OBJECTIVES

After completing this section, students should be able to manage messages and the message broker.

RABBITMQ OVERVIEW

OpenStack software provides a collection of services covering all the functionality associated with a private cloud solution. Those services themselves are built from several components, allowing for flexible and scalable configuration. OpenStack services base their back end on two functions: a database for persistence, and a message broker for supporting communications among the components of each service. Any message broker solution supporting AMQP can be used as a back end. Red Hat includes RabbitMQ as the message broker in its OpenStack product, because this provides enterprise-level features useful for setting up advanced configurations.

The following table provides some common RabbitMQ terms and definitions.

TERM	DESCRIPTION
Binding	The key or filter parameter used to route from an exchange to a queue.
Consumer	Applications that use the received messages.
Exchange	Routes producer-published messages to queues using message metadata.
Publisher/Producer	Applications that publish messages.
Queues	Stores messages until consumed by applications.
Routing key	Producer-specified message metadata used by the exchange for message routing.
Vhost	A logical division of a RabbitMQ instance, used to segregate applications, queues, and exchanges.

Message brokers send and receive messages between producer and consumer applications. Internally, RabbitmQ performs this communication using exchanges and queues, and bindings between them. When an application produces a message to send to one or more consumer applications, it places the message on an exchange to which one or more queues are bound. The exchange routes the message to the relevant queues. Consumers subscribe to the queues to receive the producer's message. Routing is based on a routing key included in the transmitted message.

MESSAGE BROKER EXCHANGE CONCEPTS

The exchange's interaction with a queue is based on the match between the routing key included in the message and the binding key associated with the queue on the related exchange. Depending on the usage of those two elements, there are several types of exchanges in RabbitMQ.

Direct

Consumers are subscribed to a queue with an associated binding key, and the producer sets the routing key of the message to be the same as that of the binding key of the queue to which the desired consumer is subscribed.

Topic

Consumers are subscribed to a queue that has a binding key that may include wildcard characters, so producers can send messages with different but related routing keys to that queue.

Fanout

Messages are broadcast to all the subscribed queues without regard for whether the routing and binding keys match.

Headers

This exchange type makes use of the message headers to perform the match against the binding arguments of the queue. This is similar to a topic exchange except that there can be multiple key-value headers, and the queue can be matched against any or all.

Message Broker Patterns Structures in OpenStack

Exchanges, queues, and bindings are combined in patterns to implement inter- and intra-application communication. The following examples describe some common programming configurations, which may be encountered when troubleshooting message broker communications.

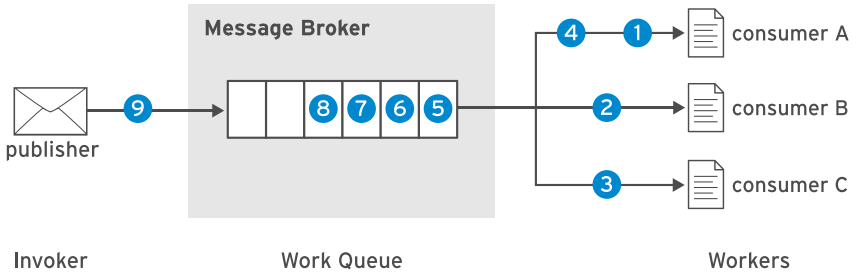


Figure 2.1: A work queue distributes messages to worker consumers

Even without using an exchange, queues are useful for handing out work to multiple consumer processes. The work queue design in Figure 2.1 distributes requests on a first come, first served basis. A single message is consumed once, unlike the fanout exchange where a message is copied to multiple queues so that all consumers receive the same messages. As long as each request message is processed, the publisher is not concerned with which worker has handled it.

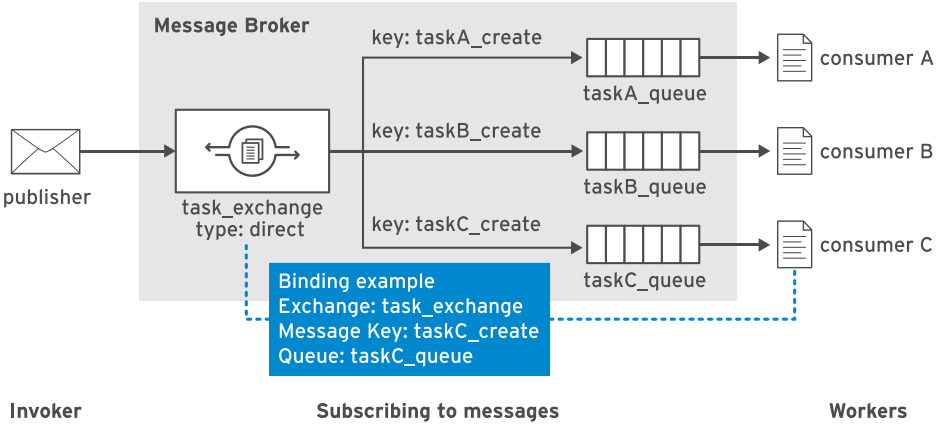


Figure 2.2: Exchanges and queues use bindings to establish routing connections

Unique tasks typically must be routed to a specific consumer application that can handle the request. Exchanges are configured to understanding message keys and correlate them to specific named queues by creating bindings between the exchange and the queue. In Figure 2.2, `taskC_create` messages will be routed to the queue named `taskC_queue`.

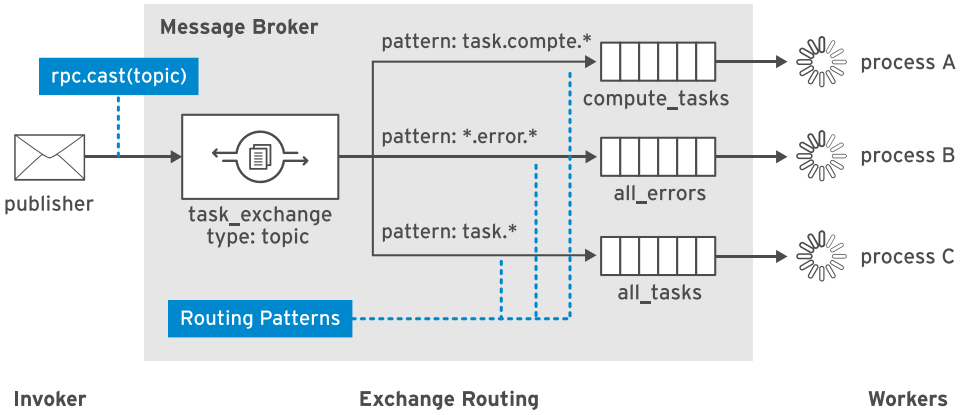


Figure 2.3: Exchanges route topic messages using user-specified pattern-matching

Exchange routing can be explicit, using pattern matching filters on topic metadata specified in the message. Using wildcards in the routing patterns, messages can be sent to one or more queues simultaneously. In Figure 2.3, only compute tasks will route to the queue named `compute_tasks`, but all tasks, including compute, will route to the queue named `all_tasks`.

Implementing RPC Using Message Queues

Individual message queues are a one-way communication from publisher to consumer. To implement a request and response communication, emulating common RPC protocol programming, necessitates using a shared, named queue to deliver the response.

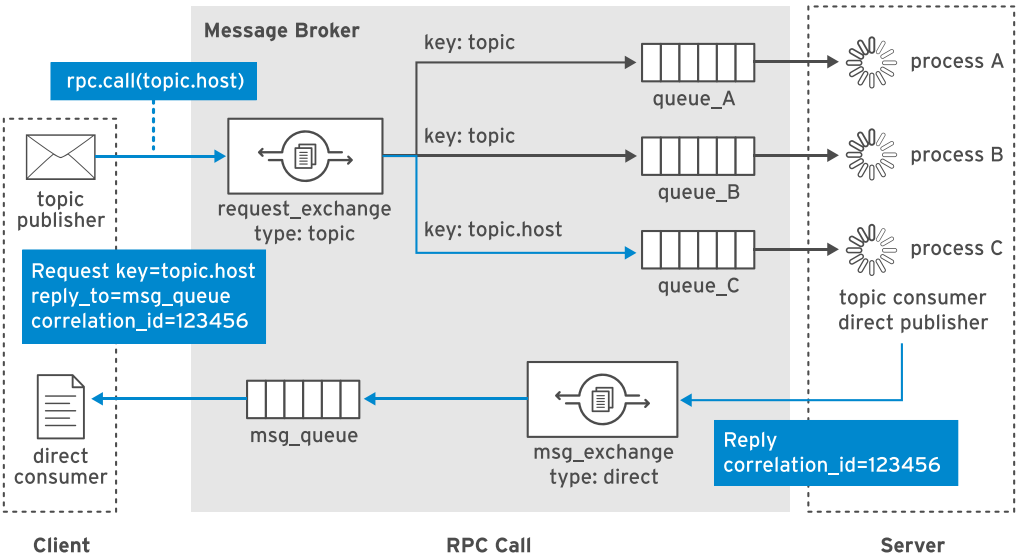


Figure 2.4: RPC calls are implemented using a pre-set direct exchange for the RPC response

When a process needs a reply to a request, it publishes a message that includes the named queue for the recipient to use to respond. The requesting process then subscribes to that queue and continues to poll for the response until it is recieved. The topic consumer worker on the server side processes the original equest, then instantiates a publisher to create the response.

OSLO MESSAGING LIBRARY

In the past, developers working on OpenStack components required some knowledge of AMQP to perform RPC operations. This drove the need for a generic interface for RPC over a messaging back end, which resulted in the Oslo Messaging Library. The `oslo.messaging.rpc` API allows developers to work on component integration without requiring any AMQP knowledge, and the extra layer of abstraction allows for the addition of non-AMQP back ends in the future.

CONFIGURATION AND LOG FILES

The following list describes the RabbitMQ configuration and log files. The live configuration files are located inside the RabbitMQ container, no longer accessed directly on the controller node. The log files are located on the controller node itself.

/etc/rabbitmq/enabled_plugins
: Contains a list of the enabled plug-ins.

/etc/rabbitmq/rabbitmq-env.conf
: Overrides the defaults built in to the RabbitMQ startup scripts.

/etc/rabbitmq/rabbitmq.config
: Provides the standard Erlang configuration file that allows the RabbitMQ core application, Erlang services, and RabbitMQ plug-ins to be configured.

/var/log/containers/rabbitmq/rabbit@overcloud-controller-0.log
: Contains logs of runtime events.

TROUBLESHOOTING SERVICE COMMUNICATION OVER MESSAGING

OpenStack services follow a component architecture. The functionalities of a service are split into different components, and each component communicates with other components using the message broker. In order to troubleshoot a problem with an OpenStack service, it is important to understand the workflow a request follows as it moves through the different components of the service. Generally, the OpenStack service architecture provides a unique component to make each service's API available. The Cinder block storage service, for example, is managed by the `cinder-api` service. The API component is the entry point to the rest of the component architecture of its service. When trying to isolate a problem with a service, check its API provider first.

After the API component has been verified, and if no errors appear in the log files, confirm that the remaining components can communicate without issue. Any error related to the RabbitMQ message broker, or its configuration in the related service configuration file, should appear in the service log files. For the Cinder block storage service, after the `cinder-api` service has processed the petition through the Cinder API, the petition is processed by both the `cinder-volume` and `cinder-scheduler` processes. These components take care of communicating among themselves using the RabbitMQ message broker to create the volume on the most feasible storage back-end location. Cinder block storage service components (`cinder-scheduler` for example) do not function correctly with a RabbitMQ back end that crashes unexpectedly. Debug the issue by checking the component-related logs, such as **`/var/log/cinder/scheduler.log`** and **`/var/log/containers/cinder/scheduler.log`** for Systemd unit-based services and Docker container-based services respectively. Then, check for problems with the component as a client of the RabbitMQ message broker. When a component crashes from RabbitMQ-related issues, it is usually due to a misconfiguration of either authorization or encryption. These errors are in a related configuration file, such as **`/etc/cinder/cinder.conf`** and for Cinder components. Sometimes, however, a crash occurs for reasons other than RabbitMQ, such as unavailable Cinder block storage services.

Commands for rabbitmqctl

Because OpenStack is now containerized, all of the RabbitMQ commands must be executed from within the container itself. The following is a list of typical commands that are used with the **rabbitmqctl** command.

- Use the **report** command to show a summary of the current status of the RabbitMQ daemon, including the number and types of exchanges and queues.

```
[user@demo ~]$ docker container exec rabbitmq-bundle-docker-0 rabbitmqctl report
...output omitted...
{listeners,
     [{clustering,25672,"::"},{amqp,5672,"172.24.1.1"},{http,15672,"::"}]},
...output omitted...
```

- Use the **list_users** command to list the RabbitMQ users.

```
[user@demo ~]$ docker container exec rabbitmq-bundle-docker-0 rabbitmqctl
 list_users
Listing users
guest   [administrator]
```

- Use the **list_exchanges** command with **rabbitmqctl** to show the default configured exchanges on the RabbitMQ daemon.

```
[user@demo ~]$ docker container exec rabbitmq-bundle-docker-0 rabbitmqctl
 list_exchanges
Listing exchanges
neutron-vo-SecurityGroup-1.0_fanout fanout
cinder-scheduler_fanout fanout
neutron-vo-SecurityGroupRule-1.0_fanout fanout
heat-engine-listener_fanout fanout
amq.topic   topic
dns topic
aodh    topic
q-plugin_fanout fanout
octavia-rpc_fanout  fanout
...output omitted...
```

- Use the **list_queues** command to list the available queues and their attributes.

```
[user@demo ~]$ docker container exec rabbitmq-bundle-docker-0 rabbitmqctl
 list_queues
Listing queues
engine  0
q-plugin    0
alarming.sample 0
q-plugin.controller0.overcloud.example.com  0
compute_fanout_ca85f4d3f7d94eb1870d00f4a0ebe08a 0
notifications.info  0
...output omitted...
```

- Use the **list_consumers** command to list all the consumers and the queues to which they are subscribed.

```
[user@demo ~]$ docker container exec rabbitmq-bundle-docker-0 rabbitmqctl
 list_consumers
Listing consumers
engine  <rabbit@controller0.2.919.0>    1   true    0   []
q-plugin    <rabbit@controller0.2.1585.0>   1   true    0   []
alarming.sample <rabbit@controller0.2.760.0>    1   true    0   []
q-plugin.controller0.overcloud.example.com  <rabbit@controller0.2.1585.0>   2
 true    0   []
compute_fanout_ca85f4d3f7d94eb1870d00f4a0ebe08a <rabbit@controller0.2.1782.0>   3
  true    0   []
...output omitted...
```

TRACING RABBITMQ MESSAGES

RabbitMQ has a built-in function to trace all messages called the Firehose Tracer. When enabled, all messages entering the system are duplicated to the `amq.rabbitmq.trace` exchange. If you create a queue, these messages can be consumed for analysis. Tracing adds load to the system so ensure that you disable tracing when your investigation is complete.

Enable and disable tracing using the **`rabbitmqctl trace_on`** and **`rabbitmqctl trace_off`** commands.

The following message was the result of a user creating an instance. It is one of many that are required to coordinate the various services to perform the required work. The message below has a routing key of **`publish.nova`** and contains **`conductor`** in the headers. This is the Nova Conductor service requesting the creation of the `finance-server2` instance.

There is a lot of information included in the message, however for troubleshooting purposes there are some key fields that are useful.

- The **`_context_request_id`** value can be used to search messages related to this request.
- The **`_reply_q`** value defines the queue that the requester will accept responses in.

In the message you can see the endpoints of all the services required to fulfill this request, as well as the information pertaining to the instance such as the key pair and the flavor.

```
{
    "_context_domain": null,
    "_msg_id": "eb26d8472f7b4a01bb7d5c3e7338cd3a",
    "_context_global_request_id": null,
    "_context_quota_class": null,
    "_context_read_only": false,
    "_context_request_id": "req-abad3917-a477-4740-8a42-a4b4d0d96f1f",
    "_context_service_catalog": [
        {
            "endpoints": [
                {
                    "adminURL": "http://172.24.1.50:8776/v2/7943c21193ae4a9d88
                                 38e69abf977442",
                    "region": "regionOne",
                    "internalURL": "http://172.24.1.50:8776/v2/7943c21193ae4a9
                                    d8838e69abf977442",
                    "publicURL": "http://172.25.250.50:8776/v2/7943c21193ae4a9
                                  d8838e69abf977442"
                }
```

```
            ],
            "type": "volumev2",
            "name": "cinderv2"
        },
        {
            "endpoints": [
                {
                    "adminURL": "http://172.24.1.50:9292",
                    "region": "regionOne",
                    "internalURL": "http://172.24.1.50:9292",
                    "publicURL": "http://172.25.250.50:9292"
                }
            ],
            "type": "image",
            "name": "glance"
        },
        {
            "endpoints": [
                {
                    "adminURL": "http://172.24.1.50:8778/placement",
                    "region": "regionOne",
                    "internalURL": "http://172.24.1.50:8778/placement",
                    "publicURL": "http://172.25.250.50:8778/placement"
                }
            ],
            "type": "placement",
            "name": "placement"
        },
        {
            "endpoints": [
                {
                    "adminURL": "http://172.24.1.50:9696",
                    "region": "regionOne",
                    "internalURL": "http://172.24.1.50:9696",
                    "publicURL": "http://172.25.250.50:9696"
                }
            ],
            "type": "network",
            "name": "neutron"
        },
        {
            "endpoints": [
                {
                    "adminURL": "http://172.24.1.50:8776/v3/7943c21193ae4a9d88
                                 38e69abf977442",
                    "region": "regionOne",
                    "internalURL": "http://172.24.1.50:8776/v3/7943c21193ae4a9
                                 d8838e69abf977442",
                    "publicURL": "http://172.25.250.50:8776/v3/7943c21193ae4a9
                                  d8838e69abf977442"
                }
            ],
            "type": "volumev3",
            "name": "cinderv3"
        }
    ],
```

```
    "args": {
        "objmethod": "save",
        "args": [],
        "objinst": {
            "nova_object.version": "2.3",
            "nova_object.changes": [
                "vm_state",
                "task_state",
                "ec2_ids"
            ],
            "nova_object.name": "Instance",
            "nova_object.data": {
                "vm_state": "building",
                "keypairs": {
                    "nova_object.version": "1.3",
                    "nova_object.name": "KeyPairList",
                    "nova_object.data": {
                        "objects": [
                            {
                                "nova_object.version": "1.4",
                                "nova_object.name": "KeyPair",
                                "nova_object.data": {
                                    "public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAAD
                                                   AQABAAABAQDHufCmzG95rJkXHZOe
                                                   +7rHMaf3me7geAfdYAc2fuoKjSVo
                                                   Bni4aF4MYkSmb1UYyXjtQ++x2+i1
                                                   3+Osn9FJZmvda0maqT6DuIcpiAxl
                                                   dVrnNJFv5L1VFGiFDbUglThPc25Y
                                                   tn7bWqg02pFJz4Nc9vN+PzVETevL
                                                   8b0tMvWPQ44MRuXCCM+UaHO1mBD2
                                                   pcEnpQ1R/MzYxTzzdvjP5iBn4GAp
                                                   7KjUw/+FvBhlNiKsXJjQGl6MHbCZ
                                                   htgsntJzl7tKGY4SgJXZaUD0TnpP
                                                   eCBlEmWNUz4hAoVMfiiZA4fLAmJ7
                                                   yZqwRcr4EVnqbmZC6CfIhVxb1J69
                                                   UHhK62KS6513MP3v Generated-b
                                                   y-Nova\\n",
                                    "user_id": "74a4cbb8304b4b359369716768963a96",
                                    "name": "example-keypair",
                                    "deleted": false,
                                    "created_at": "2018-08-14T08:30:34Z",
                                    "updated_at": null,
                                    "fingerprint":
"31:b5:bb:2c:a0:e7:39:5d:5f:5e:0e:20:8b:a5:c2:1f",
                                    "deleted_at": null,
                                    "type": "ssh",
                                    "id": 2
                                },
                                "nova_object.namespace": "nova"
                            }
                        ]
                    },
                    "nova_object.namespace": "nova"
                },
                "pci_requests": {
                    "nova_object.version": "1.1",
```

```
                "nova_object.name": "InstancePCIRequests",
                "nova_object.data": {
                    "instance_uuid": "5796ee63-adb9-4e79-ae2a-46e2215e2b86",
                    "requests": []
                },
                "nova_object.namespace": "nova"
            },
            "availability_zone": "nova",
            "terminated_at": null,
            "ephemeral_gb": 0,
            "old_flavor": null,
            "updated_at": "2018-08-15T02:54:15Z",
            "numa_topology": null,
            "ec2_ids": {
                "nova_object.version": "1.0",
                "nova_object.changes": [
                    "instance_id",
                    "kernel_id",
                    "ami_id",
                    "ramdisk_id"
                ],
                "nova_object.name": "EC2Ids",
                "nova_object.data": {
                    "instance_id": "i-00000003",
                    "kernel_id": null,
                    "ami_id": "ami-00000001",
                    "ramdisk_id": null
                },
                "nova_object.namespace": "nova"
            },
            "cleaned": false,
            "vm_mode": null,
            "flavor": {
                "nova_object.version": "1.2",
                "nova_object.name": "Flavor",
                "nova_object.data": {
                    "disabled": false,
                    "root_gb": 10,
                    "description": null,
                    "flavorid": "497a7dcc-2b13-4b6a-b8c9-b3974cb3d338",
                    "deleted": false,
                    "created_at": "2018-08-14T08:30:44Z",
                    "ephemeral_gb": 0,
                    "updated_at": null,
                    "memory_mb": 2048,
                    "vcpus": 2,
                    "extra_specs": {},
                    "swap": 0,
                    "rxtx_factor": 1.0,
                    "is_public": true,
                    "deleted_at": null,
                    "vcpu_weight": 0,
                    "id": 2,
                    "name": "m1.web"
                },
                "nova_object.namespace": "nova"
```

```
            },
            "deleted_at": null,
            "reservation_id": "r-3z094clu",
            "id": 3,
            "security_groups": {
                "nova_object.version": "1.1",
                "nova_object.name": "SecurityGroupList",
                "nova_object.data": {
                    "objects": []
                },
                "nova_object.namespace": "nova"
            },
            "disable_terminate": false,
            "root_device_name": "/dev/vda",
            "user_id": "74a4cbb8304b4b359369716768963a96",
            "uuid": "5796ee63-adb9-4e79-ae2a-46e2215e2b86",
            "default_swap_device": null,
            "info_cache": {
                "nova_object.version": "1.5",
                "nova_object.name": "InstanceInfoCache",
                "nova_object.data": {
                    "instance_uuid": "5796ee63-adb9-4e79-ae2a-46e2215e2b86",
                    "deleted": false,
                    "created_at": "2018-08-15T02:54:14Z",
                    "updated_at": null,
                    "network_info": "[]",
                    "deleted_at": null
                },
                "nova_object.namespace": "nova"
            },
            "hostname": "finance-server2",
            "launched_on": "compute1.overcloud.example.com",
            "display_description": "finance-server2",
            "key_data": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDHufCmzG95rJk
                         XHZOe+7rHMaf3me7geAfdYAc2fuoKjSVoBni4aF4MYkSmb1UYyX
                         jtQ++x2+i13+Osn9FJZmvda0maqT6DuIcpiAxldVrnNJFv5L1VF
                         GiFDbUglThPc25Ytn7bWqg02pFJz4Nc9vN+PzVETevL8b0tMvWP
                         Q44MRuXCCM+UaHO1mBD2pcEnpQ1R/MzYxTzzdvjP5iBn4GAp7Kj
                         Uw/+FvBhlNiKsXJjQGl6MHbCZhtgsntJzl7tKGY4SgJXZaUD0Tn
                         pPeCBlEmWNUz4hAoVMfiiZA4fLAmJ7yZqwRcr4EVnqbmZC6CfIh
                         Vxb1J69UHhK62KS6513MP3v Generated-by-Nova\\n",
            "deleted": false,
            "power_state": 0,
            "key_name": "example-keypair",
            "default_ephemeral_device": null,
            "progress": 0,
            "created_at": "2018-08-15T02:54:13Z",
            "project_id": "7943c21193ae4a9d8838e69abf977442",
            "launched_at": null,
            "config_drive": "",
            "node": "compute1.overcloud.example.com",
            "ramdisk_id": "",
            "access_ip_v6": null,
            "access_ip_v4": null,
            "vcpu_model": null,
            "device_metadata": null,
```

```
                "user_data": null,
                "host": "compute1.overcloud.example.com",
                "root_gb": 10,
                "display_name": "finance-server2",
                "system_metadata": {
                    "image_disk_format": "qcow2",
                    "boot_roles": "_member_",
                    "owner_user_name": "developer1",
                    "owner_project_name": "finance",
                    "image_container_format": "bare",
                    "image_min_ram": "2048",
                    "image_min_disk": "10",
                    "image_base_image_ref": "3ecec838-0c68-4fd6-ac14-94a59a69601b"
                },
                "task_state": "spawning",
                "shutdown_terminate": false,
                "tags": {
                    "nova_object.version": "1.1",
                    "nova_object.name": "TagList",
                    "nova_object.data": {
                        "objects": []
                    },
                    "nova_object.namespace": "nova"
                },
                "cell_name": null,
                "kernel_id": "",
                "ephemeral_key_uuid": null,
                "locked": false,
                "instance_type_id": 2,
                "locked_by": null,
                "launch_index": 0,
                "memory_mb": 2048,
                "vcpus": 2,
                "image_ref": "3ecec838-0c68-4fd6-ac14-94a59a69601b",
                "architecture": null,
                "auto_disk_config": false,
                "os_type": null,
                "metadata": {},
                "new_flavor": null
            },
            "nova_object.namespace": "nova"
        },
        "kwargs": {
            "expected_task_state": "block_device_mapping"
        }
    },
    "_unique_id": "5249ecdbe5a148aca72ffa11e4a265a2",
    "_context_resource_uuid": null,
    "_context_instance_lock_checked": false,
    "_context_is_admin_project": true,
    "_context_user": "74a4cbb8304b4b359369716768963a96",
    "_context_user_id": "74a4cbb8304b4b359369716768963a96",
    "_context_project_name": "finance",
    "_context_read_deleted": "no",
    "_context_user_identity": "74a4cbb8304b4b359369716768963a96 7943c21193ae4a9d8
                              838e69abf977442 - default default",
```

```
    "_reply_q": "reply_873ad7bc480243148fcf0758ba7aa539",
    "_context_auth_token": "gAAAAABbc5XTRm9IRKgBHF5WudFxFS15QZvX_uGg-tW9AAqfcWEnP
                            NWlKOiboUlxXGNJTc4m4xeMy4V7nlDM4Xtnz3EWDqqMEMNwv3JLfE
                            8I5b8wUEWYjZccnBWz1PzQ7VL8pWsGEUbJyBRZ1pb39N7o3nXRLeU
                            _IHruRvCtOqHMd5xU7Bx5vv0",
    "_context_show_deleted": false,
    "_context_tenant": "7943c21193ae4a9d8838e69abf977442",
    "_context_roles": [
        "_member_"
    ],
    "_context_is_admin": false,
    "version": "3.0",
    "_context_project_id": "7943c21193ae4a9d8838e69abf977442",
    "_context_project_domain": "default",
    "_context_timestamp": "2018-08-15T02:54:12.720169",
    "_context_user_domain": "default",
    "_context_user_name": "developer1",
    "method": "object_action",
    "_context_project": "7943c21193ae4a9d8838e69abf977442",
    "_context_remote_address": "172.24.1.1"
}
```


The following message has been truncated but shows the Neutron port binding request.


```
...output omitted...
    "args": {
        "resource_list": [
            {
                "versioned_object.version": "1.1",
                "versioned_object.name": "Port",
                "versioned_object.data": {
                    "allowed_address_pairs": [],
                    "binding": {
                        "versioned_object.version": "1.0",
                        "versioned_object.name": "PortBinding",
                        "versioned_object.data": {
                            "profile": "{}",
                            "status": "ACTIVE",
                            "vif_type": "unbound",
                            "vif_details": null,
                            "vnic_type": "normal",
                            "host": "compute1.overcloud.example.com",
                            "port_id": "68e0ba94-fec2-453c-8fc8-32ffa259e097"
                        },
                        "versioned_object.namespace": "versionedobjects"
                    },
                    "updated_at": "2018-08-15T02:54:17Z",
                    "device_owner": "compute:nova",
                    "revision_number": 8,
                    "fixed_ips": [
                        {
                            "versioned_object.version": "1.0",
                            "versioned_object.name": "IPAllocation",
                            "versioned_object.data": {
                                "subnet_id": "9f3dd8c7-b92e-4a76-b87a-
a4ffb7e9053a",
```

```
                                "network_id": "fb9a8869-7dd8-4b4e-8b54-
cd170d1c2b8a",
                                "ip_address": "192.168.1.9",
                                "port_id": "68e0ba94-fec2-453c-8fc8-32ffa259e097"
                            },
                            "versioned_object.namespace": "versionedobjects"
                        }
                    ],
                    "id": "68e0ba94-fec2-453c-8fc8-32ffa259e097",
                    "description": "",
                    "name": "",
                    "dns": null,
                    "mac_address": "fa:16:3e:d3:0b:de",
                    "project_id": "7943c21193ae4a9d8838e69abf977442",
                    "dhcp_options": [],
                    "status": "DOWN",
                    "binding_levels": [],
                    "security_group_ids": [
                        "5caab62d-1713-4e17-9da5-194566f50644"
                    ],
                    "device_id": "5796ee63-adb9-4e79-ae2a-46e2215e2b86",
                    "qos_policy_id": null,
                    "admin_state_up": true,
                    "network_id": "fb9a8869-7dd8-4b4e-8b54-cd170d1c2b8a",
                    "created_at": "2018-08-15T02:54:16Z",
                    "distributed_binding": null,
                    "security": {
                        "versioned_object.version": "1.0",
                        "versioned_object.name": "PortSecurity",
                        "versioned_object.data": {
                            "port_security_enabled": true,
                            "id": "68e0ba94-fec2-453c-8fc8-32ffa259e097"
                        },
                        "versioned_object.namespace": "versionedobjects"
                    }
                },
                "versioned_object.namespace": "versionedobjects"
            }
        ],
        "event_type": "updated"
    },
...output omitted...
```


This message shows that the members of a security group have been updated.


```
...output omitted...
    "_context_project_domain": "default",
    "_context_timestamp": "2018-08-15 02:54:17.287086",
    "method": "security_groups_member_updated",
    "_context_project": "91cb3901588c492eb5f72bb253db884f",
    "_context_roles": [
        "admin"
    ],
    "args": {
        "security_groups": [
           "5caab62d-1713-4e17-9da5-194566f50644"
```

```
        ]
    },
    "_unique_id": "8816f29b77804689b008f256e3a7c524",
    "_context_tenant_id": "91cb3901588c492eb5f72bb253db884f",
    "_context_is_admin_project": true,
...output omitted...
```

REFERENCES

Management CLI
https://www.rabbitmq.com/management-cli.html

Management Plugins
https://www.rabbitmq.com/management.html

Troubleshooting
https://www.rabbitmq.com/troubleshooting.html

DESCRIBING SERVICE COMMUNICATION BY MESSAGE BROKER

In this exercise you will enable tracing in RabbitMQ, then monitor selected components during an instance deployment.

OUTCOMES

You should be able to:

- Enable and disable tracing in RabbitMQ.
- Locate messages related to specific tasks or components.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab controlplane-message setup`** command. This ensures that all requirements are present for deploying an instance.

```
[student@workstation ~]$ lab controlplane-message setup
```

1. From `workstation`, use SSH to connect to `controller0`. Use the **`sudo -i`** command to become the `root` user.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

2. Use the **`docker exec`** and **`rabbitmqctl`** commands to create the `tracer` user in RabbitMQ, with a password of `redhat`.

```
[root@controller0 ~]# docker exec -t rabbitmq-bundle-docker-0 rabbitmqctl \
add_user tracer redhat
Creating user "tracer"
```

3. Configure permissions for the `tracer` user. Use wildcard syntax to assign all resources to each of the three permissions for **`configure`**, **`write`**, and **`read`**.

```
[root@controller0 ~]# docker exec -t rabbitmq-bundle-docker-0 rabbitmqctl \
set_permissions tracer ".*" ".*" ".*"
Setting permissions for user "tracer" in vhost "/"
```

4. Use the **`docker exec`** and **`rabbitmqctl trace_on`** commands to enable tracing in RabbitMQ.

```
[root@controller0 ~]# docker exec -t rabbitmq-bundle-docker-0 rabbitmqctl trace_on
Starting tracing for vhost "/"
```

5. Determine the interface that RabbitMQ is listening on. The default port for RabbitMQ is 5672.

```
[root@controller0 ~]# ss -tnlp | grep :5672
LISTEN   0   128   172.24.1.1:5672   *:*    users:(("beam.smp",pid=12730,fd=56))
```

6. A python script has been provided that consumes messages from the `amq.rabbitmq.trace` exchange. Use the **`-u`** and **`-p`** options to specify the `tracer` user and password. Use the **`-t`** option to provide the target IP address located in the previous step. Redirect the output to **`/tmp/rabbit.trace`** for analysis.

```
[root@controller0 ~]# ~/rmq_trace.py -u tracer -p redhat -t 172.24.1.1 \
 > /tmp/rabbit.trace
```

7. On `workstation`, open a second terminal. Authenticate to OpenStack as the `developer1` user in the `finance` project. Launch an instance using the default resources available, and wait for the deployment to finish.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--security-group default \
--key-name example-keypair \
--nic net-id=finance-network1 \
finance-server2 --wait
...output omitted...
```

8. In the first terminal, cancel the **`rmq_trace.py`** command. Use the **`docker exec`** and **`rabbitmqctl`** commands to disable tracing.

```
[root@controller0 ~]# docker exec -t rabbitmq-bundle-docker-0 rabbitmqctl \
trace_off
Stopping tracing for vhost "/"
```

9. Search through the **`/tmp/rabbit.trace`** trace log file for **`finance-server2`**.

   The first result has a routing key of `publish.nova` and has `conductor` in the headers. The entire message is in JSON format, however subsections can be taken and reformatted for readability using the **`echo '<string>' | jq .`** command. The following subsection shows the instance flavor.

```
{
    "disabled": false,
    "root_gb": 10,
    "description": null,
    "flavorid": "26df78d4-9aa7-4e57-9cdb-9e96503b82e2",
```

```
    "deleted": false,
    "created_at": "2018-08-20T05:27:03Z",
    "ephemeral_gb": 0,
    "updated_at": null,
    "memory_mb": 2048,
    "vcpus": 2,
    "extra_specs": {},
    "swap": 0,
    "rxtx_factor": 1.0,
    "is_public": true,
    "deleted_at": null,
    "vcpu_weight": 0,
    "id": 2,
    "name": "default"
}
```


In other parts of the payload, individual values are visible, such as:

```
"vm_state": "building"
```

Here you can see the key pair:


```
{
    "public_key": "ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDHufCmzG95rJkX
                           HZOe+7rHMaf3me7geAfdYAc2fuoKjSVoBni4aF4MYkSm
                           b1UYyXjtQ++x2+i13+Osn9FJZmvda0maqT6DuIcpiAxl
                           dVrnNJFv5L1VFGiFDbUglThPc25Ytn7bWqg02pFJz4Nc
                           9vN+PzVETevL8b0tMvWPQ44MRuXCCM+UaHO1mBD2pcEn
                           pQ1R/MzYxTzzdvjP5iBn4GAp7KjUw/+FvBhlNiKsXJjQ
                           Gl6MHbCZhtgsntJzl7tKGY4SgJXZaUD0TnpPeCBlEmWN
                           Uz4hAoVMfiiZA4fLAmJ7yZqwRcr4EVnqbmZC6CfIhVxb
                           1J69UHhK62KS6513MP3v Generated-by-Nova\\n",
    "user_id": "72b24f0a94c24e34a76c7fb6f2f81637",
    "name": "example-keypair",
    "deleted": false,
    "created_at": "2018-08-20T05:26:53Z",
    "updated_at": null,
    "fingerprint": "31:b5:bb:2c:a0:e7:39:5d:5f:5e:0e:20:8b:a5:c2:1f",
    "deleted_at": null,
    "type": "ssh",
    "id": 2
}
```


10. Searching for `publish.neutron` you find a message regarding port creation.

```
"event_type": "port.create.start"
```

You can then extract the **`payload`** value and reformat it to see the port information.


```
{
    "port": {
        "network_id": "d67924dc-5df0-4f17-bd4e-53a77dc9f1fa",
        "tenant_id": "db84762b38604849903c425e046704bf",
        "device_id": "b455f85e-90f6-4591-b835-f8b885ed78d5",
```

```
        "security_groups": [
            "1202065a-24ef-4fac-90fd-f5d9b0f2aec1"
        ],
        "admin_state_up": true
    }
}
```


11. Search for **`PortBinding`** and you can see the port being bound to the instance.


```
{
    "resource_list": [
        {
            "versioned_object.version": "1.1",
            "versioned_object.name": "Port",
            "versioned_object.data": {
                "allowed_address_pairs": [],
                "binding": {
                    "versioned_object.version": "1.0",
                    "versioned_object.name": "PortBinding",
                    "versioned_object.data": {
                        "profile": "{}",
                        "status": "ACTIVE",
                        "vif_type": "unbound",
                        "vif_details": null,
                        "vnic_type": "normal",
                        "host": "",
                        "port_id": "8d9a9ac3-811e-4d32-9055-e8e1a48764f6"
                    },
                    "versioned_object.namespace": "versionedobjects"
                },
                "updated_at": "2018-08-23T01:29:28Z",
                "device_owner": "",
                "revision_number": 6,
                "fixed_ips": [
                    {
                        "versioned_object.version": "1.0",
                        "versioned_object.name": "IPAllocation",
                        "versioned_object.data": {
                            "subnet_id": "e65ea1f4-12a0-4196-9e5a-1ce1441b661d",
                            "network_id": "d67924dc-5df0-4f17-bd4e-53a77dc9f1fa",
                            "ip_address": "192.168.1.8",
                            "port_id": "8d9a9ac3-811e-4d32-9055-e8e1a48764f6"
                        },
                        "versioned_object.namespace": "versionedobjects"
                    }
                ],
                "id": "8d9a9ac3-811e-4d32-9055-e8e1a48764f6",
                "description": "",
                "name": "",
                "dns": null,
                "mac_address": "fa:16:3e:09:6a:bb",
                "project_id": "db84762b38604849903c425e046704bf",
                "dhcp_options": [],
                "status": "DOWN",
                "binding_levels": [],
```

```
                "security_group_ids": [
                    "1202065a-24ef-4fac-90fd-f5d9b0f2aec1"
                ],
                "device_id": "b455f85e-90f6-4591-b835-f8b885ed78d5",
                "qos_policy_id": null,
                "admin_state_up": true,
                "network_id": "d67924dc-5df0-4f17-bd4e-53a77dc9f1fa",
                "created_at": "2018-08-23T01:29:28Z",
                "distributed_binding": null,
                "security": {
                    "versioned_object.version": "1.0",
                    "versioned_object.name": "PortSecurity",
                    "versioned_object.data": {
                        "port_security_enabled": true,
                        "id": "8d9a9ac3-811e-4d32-9055-e8e1a48764f6"
                    },
                    "versioned_object.namespace": "versionedobjects"
                }
            },
            "versioned_object.namespace": "versionedobjects"
        }
    ],
    "event_type": "updated"
}
[root@controller0 ~]# exit
[heat-admin@controller0 ~]$ exit
[student@workstation ~]$
```

There is a large amount of information available in the messaging system that may be helpful during troubleshooting.

Cleanup

On `workstation`, run **`lab controlplane-message cleanup`** to clean up resources created for this exercise.

```
[student@workstation ~]$ lab controlplane-message cleanup
```

This concludes the guided exercise.

IDENTIFYING OVERCLOUD CONTROL PLANE SERVICES

OBJECTIVES

After completing this section, students should be able to identify the shared services running on a controller node.

SHARED SERVICES OVERVIEW

There are many services that support the larger components of Red Hat OpenStack Platform. This section will review the details and use cases for each one.

SERVICE	DESCRIPTION
MariaDB	A relational database.
Redis	An in-memory key-value data store.
memcached	A distributed memory cache.
Pacemaker	Clustering software.
Ceph MON and MDS	The Ceph cluster monitor, and Metadata Server.
NoVNC and Spice	Both provide a remote console for instances.

MARIADB

MariaDB is used in OpenStack as a general purpose SQL database for any component that needs to store information. The MariaDB Galera cluster runs on controller nodes, and is currently the most popular choice for a highly available SQL database.

REDIS

Redis is an in-memory key-value data store, which was designed as a replacement for MongoDB. It supports a publisher/subscriber model and can store several types of data structures, such as strings, lists, sets, sorted sets, and hashes. Keeping all data in memory increases performance, but Redis can also save data to disk at a configurable period. Redis was designed for performance not security, so should only be used in a controlled environment.

The Redis driver is available to any OpenStack component. Example use cases include Zaqar and Keystone.

MEMCACHED

Memcached provides an in-memory cache for ephemeral data, reducing the number of accesses to a data source. It uses hashing to ensure a relatively even distribution of objects among its instances. The loss of an instance only results in the loss of a fraction of the objects. An OpenStack service automatically removes a missing Memcached instance from its list of instances.

PACEMAKER

Pacemaker is a mature clustering solution, providing high availability for OpenStack services running on controllers. Clustered services operate in an active/passive fashion. Containerized services are also supported as Pacemaker resource bundles.

CEPH MON AND MDS

The Ceph storage cluster requires a monitoring service, and the Ceph MON daemon provides this capability. It runs on each controller node and forms a Paxos part-time parliament cluster, which provides reliable storage of the Ceph cluster configuration information. To manage the Ceph file system namespace and to support CephFS, the Ceph Metadata Server (MDS) also runs on the controller nodes.

NOVNC AND SPICE

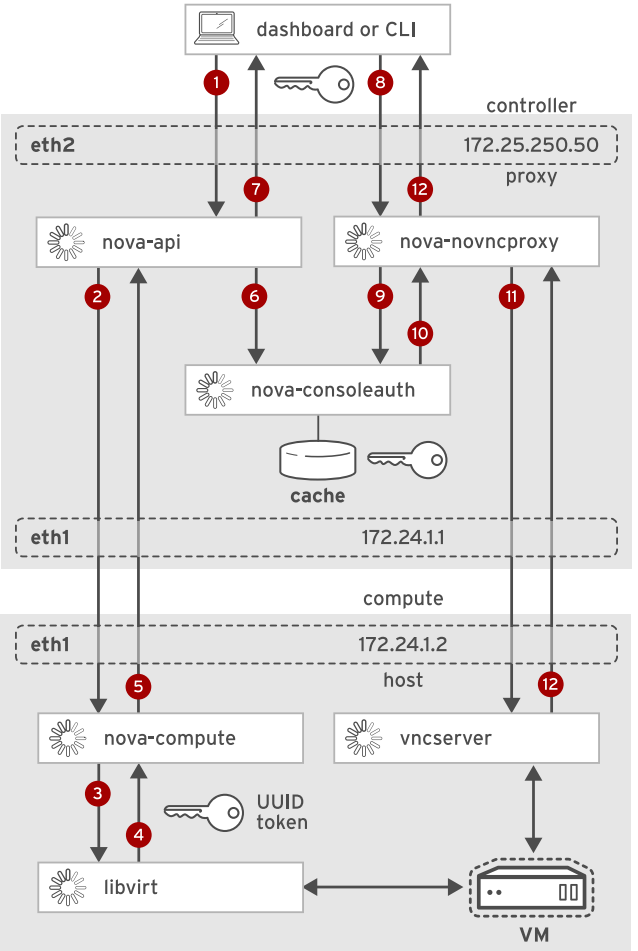
OpenStack includes and supports two protocols for providing remote virtual console access, Virtual Network Computer (VNC) and Simple Protocol for Independent Computing Environments (SPICE). VNC, the default, is a stable, mature protocol, implemented in OpenStack using the noVNC open source client. noVNC is comprised of a JavaScript library and an application that runs in desktop and mobile browsers using HTML5 and WebSockets.

Administrators may choose to use SPICE instead of VNC. SPICE is a Red Hat supported open source project designed to provide more advanced features than existing VNC clients. Although SPICE has advantages over VNC, many SPICE-HTML5 browser plugins do not support the enhanced features. To use SPICE enhancements such as multi-monitor, USB device passthrough, audio, and 3D graphics, it is recommended to use a standalone SPICE client to access OpenStack instances from the private management network.

Browser-based clients use a WebSocket proxy, either the `nova-novncproxy` (VNC) or `nova-spicehtml5proxy` (SPICE) service, to bridge remote console connections from the public network to the OpenStack management network. Both protocols use the `nova-consoleauth` shared service to authenticate client console access requests.

Using VNC to Access an Instance Console

This classroom environment uses the default VNC protocol. The VNC console access process and components are described in the following figure and text.



Each compute node runs a `vncserver` process, listening on the internal API network at one or more ports, starting at 5900 and increasing, depending on the number of instances deployed on that compute node. Each controller node runs a `novncproxy` process, listening at port 6080 on the same internal API network. The remaining services belong to the Compute Service (Nova) with components on both the controller and compute nodes.

To access the console, click the instance's name from the **dashboard Project/Compute/Instances** screen to reach the instance's detail screen. Clicking the **Console** subtab initiates a VNC console connection request. The following list, matched to the preceding graphic, describes the resulting interactions to build an instance-specific URL. Each named component is followed by its location in parentheses (node name, network name, port number):

VNC Process Flow for Accessing an Instance Console

1. A client browser (`workstation`, external), with the NoVNC plug-in, connects to `nova-api` (`controller`, internal API, port 8774) to request opening a console to a specific running server instance. The OpenStack `dashboard` connection uses `haproxy` (`controller`, external, port 80). Using the CLI invokes a non-`dashboard`browser application, and the process follows the same connection paths.

2. `Nova-api` passes a `get_vnc_console` request to `nova-compute` (`compute`, internal_API, AMQP).

3. `nova-compute` passes the `get_vnc_console` request to `libvirt` (`compute`) to locate the VM's attributes.

4. `libvirt` generates a token from the VM's UUID, and returns the token to `nova-compute` (`compute`, internal_API, AMQP). (`controller0`, internal API, AMQP).

5. `nova-compute` returns the generated token and a `connect_info` object to `nova-api` (`controller`, internal API, AMQP).

6. `nova-api` passes an `authorize_console` request to `nova-consoleauth` (`compute`, internal API, AMQP), which caches the `connect_info` object, using the token as the index, for future use when the actual connection request occurs.

7. `nova-api` also returns a `nova-novncproxy` URL and the instance-specific token to the browser (`workstation`, external).

Notice the URL at the bottom of Figure 2.5. This URL includes inline parameters for the token and instance ID for the requested server instance `demo`, at the bottom of the `Dashboard` screen as the mouse hovers over the clickable link in the blue message area.

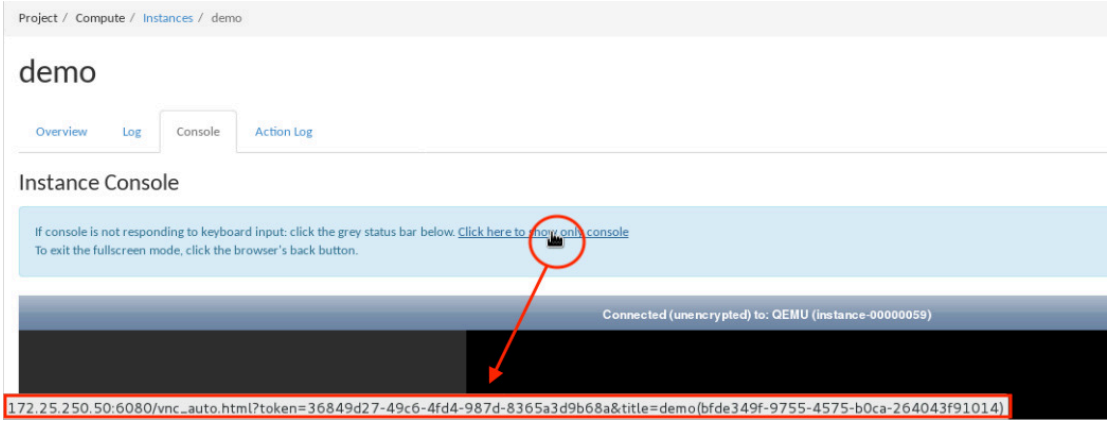


Figure 2.5: The constructed nova-novncproxy instance-specific URL

> **NOTE**
> The requirement to click the link titled Click here to show only console, plus any messages about keyboard nonresponse, is not an error. It is the result of browser settings forbidding cross-domain scripts from running automatically. You could select settings, such as *show all content* or *load unsafe scripts*, that disable protective security policies, but it is not recommended. Instead, manually click the link.

The `compute` service has previously obtained connection information for the instance and cached it with the console authorization service. The URL passed to the browser is not the direct address of the target instance console, but is the `nova-novncproxy` address, which facilitates a reverse proxy to allow the targeted instance to initiate console screen refreshes to the user's browser. The following list describes the previous graphic's remaining interactions to complete the reverse proxy VNC connection when you click the URL:

8. The browser (`workstation`, external) connects to the URL, proxied by the `dashboard`'s `haproxy` (`controller`, external, port 6080) to reach `nova-novncproxy` (`controller`, internal API, port 6080).

9. `nova-novncproxy` parses the token and instance ID from the URL and passes it to `nova-consoleauth` (`controller`, internal API, AMQP).

10. Using the token, `nova-consoleauth` retrieves the `connect_info` object from cache and returns it to `nova-novncproxy` (`controller`, internal API).

11. `nova-novncproxy` connects directly to `vncserver` (`compute`, internal API, 5900+) at the port designated for the requested VM and creates a reverse proxy configuration.

12. `nova-novncproxy` sends the console graphics back through the `dashboard haproxy` to the user's browser (`workstation`, external).

Deploying a server instance and connecting to its VNC console validates core `compute` service, `message` service, and network access functionality.

MANAGING OVERCLOUD STARTUP AND SHUTDOWN

Correctly functioning OpenStack overclouds require continuous communication and coordination between control plane services on the undercloud node and on active overcloud nodes. Once an overcloud is properly deployed and placed into service, it is expected to run perpetually. Individual nodes may be temporarily disabled for maintenance or upgrade activities, but the whole overcloud would never be shutdown, barring some unexpected or critical situation.

Regardless, OpenStack administrators should be able to properly shut down and start up an overcloud. Learning the dependency order of critical services reinforces an understanding of control plane services and infrastructure architecture. An overcloud may use a highly available (HA) controller configuration, have storage distributed across Ceph servers and hyperconverged nodes, or use custom composable roles, each of which influences the necessary command sequence. The following narrative uses this classroom's non-HA architecture as an example.

IMPORTANT
The procedure described here is intended as an educational overview. It is structured to perform a graceful shutdown for this course environment only. The exact procedure for a production OpenStack installation must be customized for that specific overcloud cluster.

WARNING
Properly shutting down an overcloud cluster takes time. Time constraints limit the ability of students to cleanly shut down and start up their OpenStack cluster for each course session. Instead, this course uses quicker but ungraceful techniques as explained in the course introduction. Classroom shortcuts in the course introduction should never be considered acceptable for a production environment.

Properly Shutting Down a Red Hat OpenStack Deployment

Red Hat OpenStack Platform is a cloud infrastructure intended for perpetual operation. Except in extraordinary circumstances, underclouds and overclouds are expected to never be completely shut down. Controller nodes are installed using high availability design, sufficient compute nodes are configured to provide redundant compute capability and storage nodes are architected for scalable redundancy and replication. Single nodes may be temporarily disabled or rebooted for maintenance without affecting the safe operation of the remainder of the OpenStack cloud.

Situations can still occur where it would be necessary to shut down a local OpenStack cluster completely. The following procedure is described in the recommended order designed to ensure that workload data is safely stored, and that services and databases are cleanly stopped so as to avoid the possiblity of service or data corruption.

Before the overcloud can be shut down, all project workloads must first be stopped. Stop (not terminate) the instances running on all 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 displayed instance UUIDs to stop each instance. In production, many applications have shutdown procedures to be run on the instance or from that application's console interface. Shutting down the server instance as shown here may be undesired.

```
[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
```

When the instances are stopped, proceed with shutting down the overcloud compute nodes. Shutting doen the compute nodes will discontinue remaining open connections to the storage nodes.

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ openstack server list -c Name -c Status -c Networks
+-------------+--------+------------------------+
| Name        | Status | Networks               |
+-------------+--------+------------------------+
| compute0    | ACTIVE | ctlplane=172.25.249.56 |
| compute1    | ACTIVE | ctlplane=172.25.249.52 |
| computehci0 | ACTIVE | ctlplane=172.25.249.54 |
| controller0 | ACTIVE | ctlplane=172.25.249.55 |
| ceph0       | ACTIVE | ctlplane=172.25.249.59 |
+-------------+--------+------------------------+
```

Log in to each compute node in turn and power it off:

```
[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.
[stack@director ~]$ ssh computehci0
[heat-admin@computehci0 ~]$ sudo poweroff
Connection to computehci0 closed by remote host.
Connection to computehci0 closed.
```

NOTE

The compute nodes will still show as **up** from the undercloud until the `compute` service synchronizes the displayed power state. This can take many minutes. If the sync routine is disabled, the status might never update. When all compute nodes are confirmed to be powered off, continue with the procedure.

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 power it off, and then return to `controller0` to stop the pacemaker cluster. Stopping the pacemaker cluster first will improperly disable admin control of Ceph servers.

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 each Ceph node and power it off:

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

The remaining OpenStack services need to be stopped to ensure system and data integrity. On each controller node, run 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)...
```

On each controller node, power them off until all are done.

```
[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 before being able to restart an overcloud.

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

In a production environment, the full OpenStack cluster is now down. Other supplementary, non-OpenStack systems may also need to be powered off.

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 nodes and services are 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 as needed for that system. 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    |
+----------------+---------+-------+
```

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 off   |
| compute0    | power off   |
| compute1    | power off   |
| computehci0 | power off   |
| ceph0       | power off   |
+-------------+-------------+
[stack@director ~]$ openstack baremetal node power on ceph0
```

Start the `controller0` node if *not* already powered on, and 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`, clear the **`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
```

When 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
[stack@director ~]$ openstack baremetal node power on computehci0
```

Start any desired project server instances that were previously shut down.

The overcloud is now fully started.

REFERENCES

Memcached
https://docs.openstack.org/ha-guide/controller-ha-memcached.html#

Ceph MON
http://docs.ceph.com/docs/master/man/8/ceph-mon/

Ceph MDS
http://docs.ceph.com/docs/master/man/8/ceph-mds/

Remote Console Access
https://docs.openstack.org/nova/queens/admin/remote-console-access.html

VIEWING OVERCLOUD CONTROL PLANE SERVICES

In this exercise, you will view overcloud control plane services.

OUTCOMES

You should be able to:

- List MariaDB databases.
- Stop and start clustered resources.
- Verify the functionality of Redis.
- Verify the functionality of Memcached.
- Connect to an instance console from the command line.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab controlplane-services setup`** command.

```
[student@workstation ~]$ lab controlplane-services setup
```

1. From `workstation`, use SSH to connect to `controller0`. Use the **`sudo -i`** command to become the `root` user.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

2. Check the status of the Ceph Monitor service. Note the details that have been called out.

```
[root@controller0 ~]# systemctl status ceph-mon@controller0.service -l -n 0
● ceph-mon@controller0.service - Ceph Monitor
   Loaded: loaded (/etc/systemd/system/ceph-mon@.service; (1)enabled; vendor
 preset: disabled)
   Active: (2)active (running) since Sun 2018-08-26 22:29:21 UTC; 4h 52min ago
  Process: 6051 (3)ExecStartPre=/usr/bin/docker rm ceph-mon-%i (code=exited,
 status=1/FAILURE)
 Main PID: 6085 (docker-current)
   CGroup: /system.slice/system-ceph\x2dmon.slice/ceph-mon@controller0.service
           └─6085 (4)/usr/bin/docker-current run --rm --name ceph-mon-controller0
 --net=host --memory=1g --cpu-quota=100000 -v /var/lib/ceph:/var/lib/ceph:z -v /
```

```
etc/ceph:/etc/ceph:z -v /var/run/ceph:/var/run/ceph:z -v /etc/localtime:/etc/
localtime:ro --net=host -e IP_VERSION=4 -e MON_IP=172.24.3.1 -e CLUSTER=ceph -e
 FSID=f9d61d02-a355-11e8-ae2b-52540001fac8 -e CEPH_PUBLIC_NETWORK=172.24.3.0/24 -e
 CEPH_DAEMON=MON 172.25.249.200:8787/rhceph/rhceph-3-rhel7:latest
```

1. The service is enabled.
2. The service is active.
3. Any existing instances are removed before starting.
4. The full command line to the process.

3. Check the status of the Ceph Metadata Server service. Note the details that have been called out.

```
[root@controller0 ~]# systemctl status ceph-mds@controller0.service -l -n 0
● ceph-mds@controller0.service - Ceph MDS
   Loaded: loaded (/etc/systemd/system/ceph-mds@.service; (1)enabled; vendor
 preset: disabled)
   Active: (2)active (running) since Sun 2018-08-26 22:29:21 UTC; 5h 33min ago
  Process: (3)6072 ExecStartPre=/usr/bin/docker rm ceph-mds-controller0
 (code=exited, status=1/FAILURE)
  Process: 6050 ExecStartPre=/usr/bin/docker stop ceph-mds-controller0
 (code=exited, status=1/FAILURE)
 Main PID: 6148 (docker-current)
   CGroup: /system.slice/system-ceph\x2dmds.slice/ceph-mds@controller0.service
           └─6148 (4)/usr/bin/docker-current run --rm --net=host --memory=4g
 --cpu-quota=100000 -v /var/lib/ceph:/var/lib/ceph:z -v /etc/ceph:/etc/ceph:z
 -v /var/run/ceph:/var/run/ceph:z -v /etc/localtime:/etc/localtime:ro -e
 CLUSTER=ceph -e CEPH_DAEMON=MDS -e MDS_NAME=controller0 --name=ceph-mds-
controller0 172.25.249.200:8787/rhceph/rhceph-3-rhel7:latest
```

1. The service is enabled.
2. The service is active.
3. Any existing instances are removed and stopped before starting.
4. The full command line to the process.

4. List the MariaDB databases to see which components are storing data in the relational database.

   4.1. Execute the **`mysql -u root -e "show databases;"`** command inside the **`galera-bundle-docker-0`** container.

```
[root@controller0 ~]# docker exec -t \
galera-bundle-docker-0 mysql -u root -e "show databases;"
+--------------------+
| Database           |
+--------------------+
| aodh               |
| cinder             |
| glance             |
| gnocchi            |
| heat               |
| information_schema |
```

```
| keystone           |
| manila             |
| mysql              |
| nova               |
| nova_api           |
| nova_cell0         |
| nova_placement     |
| octavia            |
| ovs_neutron        |
| panko              |
| performance_schema |
+--------------------+
```

4.2. As an alternative to executing commands inside the Galera container, you can query MariaDB externally.

Locate the password for the `root` MariaDB user.

```
[root@controller0 ~]# grep ^password \
/var/lib/config-data/puppet-generated/mysql/root/.my.cnf
password="3T8pj6Nu9F"
password="3T8pj6Nu9F"
```

4.3. Determine the interface that MariaDB is listening on.

```
[root@controller0 ~]# ss -tnlp | grep 3306
LISTEN  0  128  172.24.1.1:3306     *:*    users:(("mysqld",pid=16796,fd=28))
LISTEN  0  128  172.24.1.50:3306    *:*    users:(("haproxy",pid=14849,fd=26))
```

4.4. Connect to MariaDB and list the databases. Either interface will work, HAProxy will pass connections through to MariaDB.

```
[root@controller0 ~]# mysql -uroot \
-p3T8pj6Nu9F \
-h172.24.1.50 \
-e"show databases;"
+--------------------+
| Database           |
+--------------------+
| aodh               |
| cinder             |
| glance             |
| gnocchi            |
| heat               |
| information_schema |
| keystone           |
| manila             |
| mysql              |
| nova               |
| nova_api           |
| nova_cell0         |
| nova_placement     |
| octavia            |
| ovs_neutron        |
| panko              |
| performance_schema |
```

```
+--------------------+
```

5. Query Redis to ensure it is available.

   5.1. Locate the Redis password.

```
[root@controller0 ~]# docker exec -t redis-bundle-docker-0 \
grep ^requirepass /etc/redis.conf
requirepass q3W8WR4GbJaBhy64t9f2kckHD
```

   5.2. Determine the interface that Redis is listening on.

```
[root@controller0 ~]# ss -tnlp | grep redis
LISTEN  0  128  172.24.1.1:6379   *:*   users:(("redis-server",pid=17715,fd=4))
```

   5.3. Connect to Redis using the **redis-cli** command. Authenticate using the **AUTH** command and the password retrieved earlier.

```
[root@controller0 ~]# redis-cli -h 172.24.1.1
172.24.1.1:6379> AUTH q3W8WR4GbJaBhy64t9f2kckHD
OK
172.24.1.1:6379>
```

   5.4. List all keys using the **KEYS** command. Note that your results may vary. Determine the type of the `gnocchi-config` key. Use the **HGETALL** command to get all hash values of the key. Exit from the Redis CLI.

```
172.24.1.1:6379> KEYS *
 1) "_tooz_beats:controller0.0.95d6c52b-bf0b-4802-89b6-bc88e9dd0ee4"
 2) "_tooz_group:gnocchi-processing"
 3) "_tooz_group:compute-compute0.overcloud.example.com"
 4) "_tooz_group:central-global"
 5) "_tooz_group:alarm_evaluator"
 6) "_tooz_group:compute-compute1.overcloud.example.com"
 7) "_tooz_beats:controller0.0.11d314d7-8b44-45c9-a98d-5838cb682f30"
 8) "_tooz_beats:e7ebea7f-7f04-4006-a493-a96a64cc2606"
 9) "_tooz_beats:bc92a536-09cc-43c4-867a-2c472e7e144c"
10) "_tooz_groups"
11) "_tooz_beats:9c9af69a-f09a-449f-8dbb-e1b7edb63f90"
12) "gnocchi-config"
172.24.1.1:6379> TYPE gnocchi-config
hash
172.24.1.1:6379> HGETALL gnocchi-config
1) "sacks"
2) "128"
172.24.1.1:6379> exit
[root@controller0 ~]#
```

6. Examine memcached.

   6.1. Determine the interface that Memcached is listening on.

```
[root@controller0 ~]# ss -tnlp | grep memcached
```

```
LISTEN   0   128   172.24.1.1:11211    *:*    users:(("memcached",pid=4559,fd=16))
```

6.2. Execute the **memcached-tool** command inside the memcached container. By default this displays the **slab class** class statistics.

```
[root@controller0 ~]# docker exec -t memcached memcached-tool 172.24.1.1
  #  Item_Size  Max_age   Pages   Count   Full?  Evicted Evict_Time OOM
  6     304B        85s       1       1     yes        0        0    0
 23    13.6K       263s       1       3     yes        0        0    0
```

There are two **slab classes** above. Note that your results may vary. We can see the size, age, and other relevant information.

6.3. Execute **memcached-tool** again with the **stats** option.

```
[root@controller0 ~]# docker exec -t memcached memcached-tool 172.24.1.1 stats
#172.24.1.1:11211  Field       Value
         accepting_conns           1
               auth_cmds           0
             auth_errors           0
                   bytes       33823
              bytes_read      299122
           bytes_written     3123740
              cas_badval           0
                cas_hits           0
              cas_misses           0
                cmd_flush          0
                  cmd_get         305
                  cmd_set          60
               cmd_touch           0
             conn_yields           0
   connection_structures           5
   crawler_items_checked           0
       crawler_reclaimed           0
        curr_connections           4
              curr_items           4
                decr_hits          0
              decr_misses          0
              delete_hits          0
            delete_misses          0
       evicted_unfetched           0
                evictions          0
       expired_unfetched          17
             get_expired          17
             get_flushed           0
                get_hits         281
              get_misses          24
              hash_bytes      524288
       hash_is_expanding           0
        hash_power_level          16
                incr_hits          0
              incr_misses          0
                 libevent 2.0.21-stable
          limit_maxbytes  4100980736
     listen_disabled_num           0
        log_watcher_sent           0
```

```
     log_watcher_skipped           0
      log_worker_dropped           0
      log_worker_written           0
       lrutail_reflocked           0
            malloc_fails           0
                     pid           7
            pointer_size          64
               reclaimed          21
            reserved_fds          10
           rusage_system    0.201163
             rusage_user    0.080465
                 threads           2
                    time  1535415920
time_in_listen_disabled_us           0
       total_connections           6
             total_items          60
              touch_hits           0
            touch_misses           0
                  uptime        3052
                 version      1.4.39
```

There are several pieces of information here:

- The value of `curr_items` should match the total of the count column in the slab class output.
- The `cmd_set` and `cmd_get` values indicate the caching ratio. The `cmd_get` value should be higher, if not then your stored items may be expiring too quickly.
- The value of `get_misses` indicates how often an item was not present.

7. Use the **pcs** command to manage Pacemaker resources.

7.1. Examine the status of all Pacemaker resources using the **pcs resource show** command.

```
[root@controller0 ~]# pcs resource show
 Docker container: rabbitmq-bundle [172.25.249.200:8787/rhosp13/openstack-
rabbitmq:pcmklatest]
   rabbitmq-bundle-0 (ocf::heartbeat:rabbitmq-cluster): Started controller0
 Docker container: galera-bundle [172.25.249.200:8787/rhosp13/openstack-
mariadb:pcmklatest]
   galera-bundle-0 (ocf::heartbeat:galera): Master controller0
 Docker container: redis-bundle [172.25.249.200:8787/rhosp13/openstack-
redis:pcmklatest]
   redis-bundle-0 (ocf::heartbeat:redis): Master controller0
 ip-172.25.249.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.25.250.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.1.51 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.1.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.3.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.4.50 (ocf::heartbeat:IPaddr2): Started controller0
 Docker container: haproxy-bundle [172.25.249.200:8787/rhosp13/openstack-
haproxy:pcmklatest]
   haproxy-bundle-docker-0 (ocf::heartbeat:docker): Started controller0
 Docker container: ovn-dbs-bundle [172.25.249.200:8787/rhosp13/openstack-ovn-
northd:latest]
```

```
   ovn-dbs-bundle-0 (ocf::ovn:ovndb-servers): Master controller0
 Docker container: openstack-cinder-volume [172.25.249.200:8787/rhosp13/openstack-
cinder-volume:pcmklatest]
   openstack-cinder-volume-docker-0 (ocf::heartbeat:docker): Started controller0
 Docker container: openstack-manila-share [172.25.249.200:8787/rhosp13/openstack-
manila-share:pcmklatest]
   openstack-manila-share-docker-0 (ocf::heartbeat:docker): Started controller0
```

7.2. Disable the `openstack-manila-share` resource. Execute **`pcs resource show`** again to confirm.

```
[root@controller0 ~]# pcs resource disable openstack-manila-share
[root@controller0 ~]# pcs resource show
...output omitted...
 Docker container: openstack-manila-share [172.25.249.200:8787/rhosp13/openstack-
manila-share:pcmklatest]
   openstack-manila-share-docker-0 (ocf::heartbeat:docker): Stopped (disabled)
```

7.3. Enable the `openstack-manila-share` resource. Execute the **`pcs resource show openstack-manila-share`** command to view details of the resource.

```
[root@controller0 ~]# pcs resource enable openstack-manila-share
[root@controller0 ~]# pcs resource show openstack-manila-share
 Bundle: openstack-manila-share
  Docker: image=172.25.249.200:8787/rhosp13/openstack-manila-share:pcmklatest
 network=host options="--ipc=host --privileged=true --user=root --log-
driver=journald -e KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=1 run-command="/
bin/bash /usr/local/bin/kolla_start"
  Storage Mapping:
   options=ro source-dir=/var/lib/kolla/config_files/manila_share.json target-
dir=/var/lib/kolla/config_files/config.json (manila-share-cfg-files)
   options=ro source-dir=/var/lib/config-data/puppet-generated/manila/ target-
dir=/var/lib/kolla/config_files/src (manila-share-cfg-data)
   options=ro source-dir=/etc/hosts target-dir=/etc/hosts (manila-share-hosts)
   options=ro source-dir=/etc/localtime target-dir=/etc/localtime (manila-share-
localtime)
   options=rw source-dir=/dev target-dir=/dev (manila-share-dev)
   options=rw source-dir=/run target-dir=/run (manila-share-run)
   options=rw source-dir=/sys target-dir=/sys (manila-share-sys)
   options=ro source-dir=/lib/modules target-dir=/lib/modules (manila-share-lib-
modules)
   options=rw source-dir=/var/lib/manila target-dir=/var/lib/manila (manila-share-
var-lib-manila)
   options=ro source-dir=/etc/pki/ca-trust/extracted target-dir=/etc/pki/ca-trust/
extracted (manila-share-pki-extracted)
   options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-dir=/etc/pki/tls/
certs/ca-bundle.crt (manila-share-pki-ca-bundle-crt)
   options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-dir=/etc/
pki/tls/certs/ca-bundle.trust.crt (manila-share-pki-ca-bundle-trust-crt)
   options=ro source-dir=/etc/pki/tls/cert.pem target-dir=/etc/pki/tls/cert.pem
 (manila-share-pki-cert)
   options=rw source-dir=/var/log/containers/manila target-dir=/var/log/manila
 (manila-share-var-log)
   options=ro source-dir=/etc/ceph target-dir=/etc/ceph (manila-share-ceph-cfg-
dir)
[root@controller0 ~]# logout
```

```
[heat-admin@controller0 ~]$ logout
Connection to controller0 closed.
[student@workstation ~]$
```

8. Connect to the console of an instance.

8.1. Source the environment file for `developer1` in the `finance` project. Obtain the console URL for the `finance-server3` instance.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack console url show \
-c url -f value finance-server3
http://172.25.250.50:6080/vnc_auto.html?token=99635dd0-f87b-4b1b-b8c8-48b1c06a4afc
```

Note the port value of 6080 in the console URL.

8.2. Log in to `controller0` then become `root`. Display the process listening on port 6080.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# ss -tnlp | grep 6080
LISTEN  0 128 172.25.250.50:6080  *:* users:(("haproxy",pid=15285,fd=31))
LISTEN  0 128 172.24.1.50:6080    *:* users:(("haproxy",pid=15285,fd=30))
LISTEN  0 100 172.24.1.1:6080     *:* users:(("nova-novncproxy",pid=3902,fd=4))
[root@controller0 ~]# logout
[heat-admin@controller0 ~]$ logout
Connection to controller0 closed.
[student@workstation ~(developer1-finance)]$
```

Note that `nova-novncproxy` is the service we are connecting to. It is fronted by HAProxy on the external network, allowing users to connect through to the instance console on the management network.

8.3. Connect to the URL to verify the service is working.

```
[student@workstation ~(developer1-finance)]$ firefox \
http://172.25.250.50:6080/vnc_auto.html?token=99635dd0-f87b-4b1b-b8c8-48b1c06a4afc
```

Cleanup

From `workstation`, run **`lab controlplane-services cleanup`** to clean up resources created for this exercise.

```
[student@workstation ~]$ lab controlplane-services cleanup
```

This concludes the guided exercise.

MANAGING INTERNAL OPENSTACK COMMUNICATION

PERFORMANCE CHECKLIST

In this lab, you will configure the Identity service to use memcached as a caching layer for tokens.

OUTCOMES

You should be able to:

- Change the configuration of a containerized service.
- Verify the use of Memcached for token caching.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab controlplane-review setup`** command. This ensures that the environment has been properly configured for this lab.

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

1. Determine the current rate of increase for the memcached **`get_hits`** parameter.
2. Enable token caching for the Identity service on controller0. Use the settings in the table below to configure **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf`**.

SECTION	OPTION	VALUE
cache	enabled	true
cache	backend	dogpile.cache.memcached
cache	backend_argument	url:172.24.1.1:11211
catalog	caching	false
domain_config	caching	false
federation	caching	false
revoke	caching	false
role	caching	false
token	caching	true

SECTION	OPTION	VALUE
token	**cache_on_issue**	**true**
identity	**caching**	**false**

3. Use the **openstack** command to test for a sharp increase in the value for the memcached **get_hits** parameter.

Evaluation

On workstation, run the **lab controlplane-review grade** script to confirm the success of this exercise.

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

Cleanup

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

```
[student@workstation ~]$ lab controlplane-review cleanup
```

This concludes the lab.

MANAGING INTERNAL OPENSTACK COMMUNICATION

PERFORMANCE CHECKLIST

In this lab, you will configure the Identity service to use memcached as a caching layer for tokens.

OUTCOMES

You should be able to:

- Change the configuration of a containerized service.
- Verify the use of Memcached for token caching.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab controlplane-review setup`** command. This ensures that the environment has been properly configured for this lab.

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

1. Determine the current rate of increase for the memcached **`get_hits`** parameter.

   1.1. From `workstation`, log in to `controller0`, and change to the `root` user.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

   1.2. Use the **`watch`** command to monitor the *`get_hits`* parameter.

```
[root@controller0 ~]# watch -n 5 \
"memcached-tool 172.24.1.1 stats | grep -A2 get_hits"
Every 5.0s: memcached-tool 172.24.1.1 stats | grep -A2 get_hits

                 get_hits         944
               get_misses          88
               hash_bytes      524288
```

   Note that the rate changes quite slowly.

2. Enable token caching for the Identity service on controller0. Use the settings in the table below to configure **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf`**.

SECTION	OPTION	VALUE
cache	enabled	true
cache	backend	dogpile.cache.memcached
cache	backend_argument	url:172.24.1.1:11211
catalog	caching	false
domain_config	caching	false
federation	caching	false
revoke	caching	false
role	caching	false
token	caching	true
token	cache_on_issue	true
identity	caching	false

2.1. Open a new terminal on `workstation`, connect using SSH to `controller0`, then become `root`.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

2.2. Use **`crudini`** to modify the relevant options.

```
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
cache enabled true
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
cache backend dogpile.cache.memcached
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
cache backend_argument url:172.24.1.1:11211
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
catalog caching false
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
domain_config caching false
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
federation caching false
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
revoke caching false
[root@controller0 ~]# crudini --set \
```

```
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
role caching false
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
token caching true
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
token cache_on_issue true
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
identity caching false
```

2.3. Restart the `keystone` container to effect the changes.

```
[root@controller0 ~]# docker restart keystone
keystone
[root@controller0 ~]# logout
[heat-admin@controller0 ~]$ logout
[student@workstation ~]$
```

3. Use the **`openstack`** command to test for a sharp increase in the value for the memcached **`get_hits`** parameter.

3.1. In the first terminal, verify the current value of *`get_hits`*.

```
Every 5.0s: memcached-tool 172.24.1.1 stats | grep -A2 get_hits

                 get_hits        1283
               get_misses         265
               hash_bytes      524288
```

3.2. In the second terminal, source the environment file for `operator1` in the `production` project. Perform several **`openstack`** commands.

```
[student@workstation ~]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack image list -f json
...output omitted...
  {
    "Status": "active",
    "ID": "47c4a5c1-27cb-49c8-a38a-639654e8bb35",
    "Name": "rhel7"
  },
...output omitted...
[student@workstation ~(operator1-production)]$ openstack flavor list -f json
...output omitted...
  {
    "Name": "default",
    "RAM": 2048,
    "Ephemeral": 0,
    "VCPUs": 2,
    "Is Public": true,
    "Disk": 10,
    "ID": "4d1f2fe1-cb51-4e53-99b9-00334eaf8c7b"
  },
...output omitted...
[student@workstation ~(operator1-production)]$ openstack network list -f json
```

```
...output omitted...
  {
    "Subnets": "de0c65db-5d75-49a3-b834-ffafb384c4c6",
    "ID": "50060e33-2385-45b2-8a23-7fc589a31c8a",
    "Name": "production-network1"
  },
...output omitted...
[student@workstation ~(operator1-production)]$ openstack subnet list -f json
[
...output omitted...
  {
    "Network": "50060e33-2385-45b2-8a23-7fc589a31c8a",
    "Subnet": "192.168.1.0/24",
    "ID": "de0c65db-5d75-49a3-b834-ffafb384c4c6",
    "Name": "production-subnet1"
  }
]
```

3.3. In the first terminal, recheck the value of **get_hits**.

```
Every 5.0s: memcached-tool 172.24.1.1 stats | grep -A2 get_hits

                get_hits        1407
              get_misses         265
              hash_bytes      524288

[root@controller0 ~]# logout
[heat-admin@controller0 ~]$ logout
[student@workstation ~]$
```

Note that the value of **get_hits** has increased at a faster rate than previously.

Evaluation

On workstation, run the **lab controlplane-review grade** script to confirm the success of this exercise.

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

Cleanup

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

```
[student@workstation ~]$ lab controlplane-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- API endpoints are discoverable through the use of a service catalog. Endpoints are segregated by networks to avoid exposure of sensitive information.
- All Red Hat OpenStack Platform components communicate with each other using an AMQP-compliant messaging system. RabbitMQ is the default, however the use of other messaging back ends is possible. The components communicate using RPC implemented on top of the messaging system. The Oslo Messaging library abstracts the interprocess communication away from AMQP so that developers have no need to understand it.
- RabbitMQ has a tracing feature which allows administrators to view messages passing through the messaging system. Tracing adds overhead and so should only be enabled when required.
- There are many services that support the main components. They allow components to store data in a relational or nonrelational database, perform caching in memory to increase performance, and enable users to connect to instance consoles.

CHAPTER 3

INTEGRATING IDENTITY MANAGEMENT

GOAL

Configure a scalable identity service and manage scalable projects.

OBJECTIVES

- Describe the installation and architecture of a Red Hat Identity Management back end for the OpenStack identity service.
- Manage user tokens that implement user authorization for accessing OpenStack services.
- Manage project quotas, domains, hierarchies, and groups.

SECTIONS

- Managing an Integrated IdM Back-end Configuration (and Guided Exercise)
- Managing Identity Service Tokens (and Guided Exercise)
- Managing Project Organization (and Guided Exercise)

LAB

- Integrating Identity Management

MANAGING AN INTEGRATED IDM BACK-END CONFIGURATION

OBJECTIVES

After completing this section, students should be able to describe the installation and architecture of a Red Hat Identity Management back end for the OpenStack identity service.

IDENTITY SERVICE ARCHITECTURE

The OpenStack Identity Service (code-named Keystone) provides authentication, role-based authorization, policy management, and token handling using internal service functions categorized as identity, policy, token, resource, role assignment, and catalog. The Identity Service API is available at configurable endpoints segregated by public and internal traffic. Multiple controller nodes use Pacemaker with a virtual IP (VIP) to provide redundant access to the API. The internal service functions manage different aspects of the Identity Service:

Identity

Identity encompasses authentication and authorization functions. Users are a digital representation of a person, system, or service using other OpenStack services. Users are authenticated before requesting services from OpenStack components. Users must be assigned a role to participate in a project. Users may be managed using groups, introduced in Identity Service v3, which can be assigned roles and attached to projects in the same way as individual users.

Projects (previously known as *tenants*) are collections of owned resources such as networks, images, servers, and security groups. These are structured according to the development needs of an organization. A project can represent a customer, account, or any organizational unit. With Identity Service v3, projects can contain subprojects, which inherit project role assignments and quotas from parent projects.

Resource

Resource functions manage *domains*, which are an Identity Service v3 entity for creating segregated collections of users, groups, and projects. Domains allow multiple organizations to share a single OpenStack installation. Users, projects, and resources created in one domain cannot be transferred to another domain; by design, they must be recreated. OpenStack creates a single domain named `default` for a new installation. In Identity Service v2, multiple domains are not recognized and all activities use the `default` domain.

Token

Token functions create, manage, and validate time-limited tokens, which users pass to other OpenStack components to request a service. A token is a structured enumeration of user access rights designed to simplify the requirement that each individual OpenStack service request be verified for sufficient user privilege. Token protocols have evolved since the early OpenStack days, and are discussed later in this chapter.

Policy

Policy functions provide a rule-based authorization engine and an associated rule management interface. Policy rules define the capabilities of roles. Default roles include `admin`, `_member_`, `swiftoperator`, and `heat_stack_user`. You can build policies to create custom roles.

Role Assignment

Role assignment functions are used to assign users to projects. Users do not belong to projects; instead they have a role in a project. Users may be assigned multiple roles for the same project, and may also be assigned different roles in multiple projects.

Roles define a set of user privileges to perform specific operations on OpenStack services, defined by policy definitions. The most commonly recognized roles are `_member_`, which can perform all normal activities within a project, and `admin`, which adds additional permissions to create users, projects, and other restricted resource objects.

Catalog

Catalog functions store connection information about every other OpenStack service component, in the form of endpoints. The catalog contains multiple endpoint entries for each service, to allow service traffic to be segregated by public, internal, and administration tasks for traffic management and security reasons. Because OpenStack services may be redundantly installed on multiple controller and compute nodes, the catalog contains endpoints for each. When users authenticate and obtain a token to use when accessing services, they are, at the same time, being given the current URL of the requested service.

> **NOTE**
> Red Hat OpenStack Platform supports both Identity Service v2 and v3. Identity v3 requires the use of the new authentication environment variables `OS_IDENTITY_API_VERSION` and `OS_DOMAIN_NAME`, and a change to the `OS_AUTH_URL` for the new version's endpoint. This Red Hat OpenStack Administration II course only uses Identity Service v3.

Each listed Identity Service function supports multiple choices of back ends, defined through plug-ins, which can be one of the following types (not all functions support all back-end types):

- *Key Value Store*: A file-based or in-memory dictionary using primary key lookups.
- *Memcached*: A distributed-memory shared caching structure.
- *Structured Query Language*: OpenStack uses *SQLAlchemy* as the default persistent data store for most components. *SQLAlchemy* is a Python-based SQL toolkit.
- *Pluggable Authentication Module*: Using the Linux `PAM` authentication service.
- *Lightweight Directory Access Protocol*: Uses the LDAP protocol to connect to an existing back-end directory, such as IdM or AD, for user authentication and role information.

AUTHENTICATION METHODS

A key function of the Identity Service is the authentication of user and project credentials. The purpose of authentication is to validate the identity of each user. After the Identity Service has confirmed a user's identity it can proceed with granting the user authorization and access to a Red Hat OpenStack Platform environment.

Red Hat OpenStack Platform supports various authentication methods. The most basic authentication method validates user names and passwords against account information stored in a local database. In large enterprise environments, Red Hat OpenStack Platform can validate user credentials against external authentication back ends such as Active Directory (AD), Red Hat IdM, or a generic LDAP server.

INSTALLING RED HAT IDM

Red Hat IdM combines LDAP, Kerberos, DNS, and PKI protocols with a rich management framework to provide identity services to Linux clients within enterprise environments. IdM is the Red Hat tuned and supported release of the upstream open source FreeIPA project, built from popular Linux administration components including PAM, BIND DNS, MIT Kerberos, and Dogtag PKI, integrated with the core 389 LDAP Directory Server.

Because IdM is included in a standard Red Hat Enterprise Linux subscription, minimal configuration is required to use it. Before you can create your IdM instance, you need to install at least the *ipa-server* package. Depending on your architecture, if you want IdM to manage a DNS server as well, you have to install two additional packages to set up DNS. In both cases, you can use the **yum** command to install all required packages. This example shows the installation of the IdM package with the additional DNS packages:

```
[root@demo ~]# yum install ipa-server ipa-server-dns
(...)
Transaction Summary
============================================
Install  2 Packages (+227 Dependent packages)

Total download size: 113 M
Installed size: 300 M
Is this ok [y/d/N]: y
```

A substantial number of dependencies are installed with the *ipa-server* package. This includes *389-ds-base* for the LDAP service and *krb5-server* for the Kerberos service, as well as IdM tools.

The IdM Installation Script

After all of the required packages have been installed, you can create a new IdM instance. Run the **ipa-server-install** command to create this instance. This launches the script interactively. You are prompted for information to help you set up a basic IdM instance. This helps you quickly set up a basic IdM instance that does not make use of more advanced features such as DNS management or CA certification management.

The setup script creates a new instance of an IdM domain and configures all the required services and policies:

- 389 LDAP Directory Server instance
- Kerberos key distribution center (KDC)
- Active Directory WinSync plug-in
- Certificate Authority (CA)
- Network time protocol daemon (ntpd)
- Apache (httpd)
- SELinux targeted policy
- Domain Name Service (DNS). (This is optional.)

Depending on your needs for the IdM domain, the setup process can be minimal, in which you provide a small amount of information, or it can be very specific, with user-defined settings for various parts of the IdM services. If you have to create a customized IdM environment, you can pass those configuration settings as arguments to the **ipa-server-install** script. Using those arguments allows for a versatile configuration that can easily be scripted. With these noninteractive arguments, you can also configure additional IdM settings that are not used during the interactive installation mode. To access the full list of available options, consult the **ipa-server-install** man page.

RED HAT IDM INTEGRATION

To use Red Hat IdM to authenticate user credentials in a Red Hat OpenStack Platform environment, you must configure both the Red Hat IdM server and the Red Hat OpenStack Platform environment for the integration. The IdM server requires an account that allows LDAP lookups from the Identity service running in the Red Hat OpenStack Platform environment. The communication between the two environments also requires making the IdM server's CA certificate available to the Identity service. Finally, you need to configure the Identity service for multidomain support and for LDAP communication to the IdM back end.

Configuring an LDAP Look-up Account

To perform user credential authentication using a Red Hat IdM back end, the Identity service requires an LDAP look-up account on the IdM server. Run the **`ipa user-add`** command as the `admin` user to create an LDAP look-up account on the IdM server. Use the **`ipa group-add-member`** command to add the LDAP look-up account to the appropriate IdM user group.

The following example shows the creation of an account named `ldaplookup` for the purpose of performing LDAP lookups. The password for the account is set to `redhat`. The example also shows the creation of an `openstack` user group and the addition of the `ldaplookup` user into that group.

```
[root@idm ~]# kinit admin
[root@idm ~]# echo redhat | ipa user-add ldaplookup --first=OpenStack --last=LDAP
 --password
[root@idm ~]# ipa group-add --desc="OpenStack Users" openstack
[root@idm ~]# ipa group-add-member --users=ldaplookup openstack
```

Installing the IdM CA Certificate

In order for the Identity service to perform LDAP lookups against the IdM server using LDAPS, it requires the installation of the IdM server's CA certificate. By default, this CA certificate is called **`/etc/ipa/ca.crt`** on the IdM server. To make the certificate available to the Identity service, install it to the **`/etc/ssl/certs`** directory on the Identity service container. Convert the certificate to PEM format and then install the PEM version to the **`/etc/pki/ca-trust/source/anchors`** directory on the Identity service container.

For environments using a containerized Identity service, install the certificates to the appropriate directories under the service container configuration tree on the controller node. Copy the CA certificate to the **`/var/lib/config-data/puppet-generated/keystone/etc/ssl/certs`** directory on the controller. Likewise, copy the PEM version of the CA certificate to the **`/var/lib/config-data/puppet-generated/keystone/etc/pki/ca-trust/source/anchors`** directory. Create the parent directories for these certificate files if they do not already exist. Set the ownership of both the directories and the certificate files to the `42425` UID and GID to ensure that these are accessible by the `keystone` user.

Configuring the Horizon Dashboard

By default, the Identity Service utilizes a single domain named `default`. Integration with external authentication back ends requires the creation of a separate Identity domain for each back end. Therefore, to integrate with an IdM back end, you must first enable multidomain support in the Horizon dashboard so that users can log in using a specific domain.

To enable multidomain support, set the following parameters in the **`/var/lib/config-data/puppet-generated/horizon/etc/openstack-dashboard/local_settings`** configuration file.

```
OPENSTACK_API_VERSIONS = {
```

```
  "identity": 3
}
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = 'Default'
```

With all the Horizon configuration changes in place, use the **`docker horizon restart`** command to restart the container and apply those changes.

Configuring the Identity Service

Set the following parameters in the **`identity`** section of the **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf`** configuration file to enable multidomain support in the dashboard.

```
domain_specific_drivers_enabled = true
domain_config_dir = /etc/keystone/domains
```

In the same file, set the following parameter in the **`assignment`** section.

```
driver = sql
```

With multidomain support enabled in the Identity service configuration, you can now configure the Identity service to perform LDAP lookups against the IdM back end. This configuration is saved in the **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/domains`** directory on the controller node. Each external authentication back end requires a separate domain authentication configuration file in this directory. The file names of these domain authentication configuration files must conform to the **`keystone.`*`DOMAIN`*`.conf`** format, where *DOMAIN* is the name of the domain.

The following example demonstrates the contents of the domain configuration file, **`keystone.idm.conf`**, which configures the Identity service to use the `ldaplookup` IdM user account and the IdM server CA certificate installed in previous examples.

```
[ldap]
url =  ldaps://idm.lab.example.com
user = uid=ldaplookup,cn=users,cn=accounts,dc=lab,dc=example,dc=net
user_filter =
 (memberOf=cn=openstack,cn=groups,cn=accounts,dc=lab,dc=example,dc=net)
password = redhat
user_tree_dn = cn=users,cn=accounts,dc=lab,dc=example,dc=net
user_objectclass = inetUser
user_id_attribute = uid
user_name_attribute = uid
user_mail_attribute = mail
user_pass_attribute =
user_allow_create = False
user_allow_update = False
user_allow_delete = False
group_objectclass = groupofnames
group_tree_dn = cn=groups,cn=accounts,dc=lab,dc=example,dc=net
group_id_attribute = cn
group_name_attribute = cn
group_allow_create = False
group_allow_update = False
group_allow_delete = False
```

```
use_tls = False
query_scope = sub
chase_referrals = false
tls_cacertfile = /etc/ssl/certs/ca.crt

[identity]
driver = ldap
```

Create the parent directory for the domain authentication configuration files, **/var/lib/config-data/puppet-generated/keystone/etc/keystone/domains**, if it does not already exist. Set the ownership of both the directory and its contents to the 42425 UID and GID to ensure that these are accessible by the keystone user.

With all the Horizon configuration changes in place, use the **docker horizon restart** command to restart the container and apply those changes.

Creating Identity Domains

Because integration with external authentication back ends requires the creation of a separate Identity domain for each back end, you must create a new domain for the newly integrated Red Hat IdM back end. Use the **openstack domain create** command to create the domain as the admin user. The following example creates a new domain named idm.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack domain create idm
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| enabled     | True                             |
| id          | 012f8a266990444da2130c6e8c68cdd1 |
| name        | idm                              |
| tags        | []                               |
+-------------+----------------------------------+
```

To enable the OpenStack admin user access to the administer the new domain, grant it the admin role for the new domain.

```
[student@workstation ~]$ openstack role add --domain idm --user admin admin
```

With the new domain created for the IdM back end, use the **openstack user list** command to search for a list of users in the new domain and to verify the integration. The command returns a list of IdM users.

```
[student@workstation ~]$ openstack user list --domain idm
```

With the IdM integration in place, users can log in to the dashboard using their IdM credentials. When you enable multidomain support in dashboard and identity, in addition to the user name and password, the dashboard requires users to specify a domain and a project. Therefore, for IdM users to be able to access the dashboard, you must first use the **openstack role add** command to grant them a role for at least one project. The following example demonstrates granting the IdM idm_members user group the _member_ role for the research project in the idm domain.

```
[student@workstation ~]$ openstack role add \
```

```
--project research \
--project-domain idm \
--group developer2 \
--group-domain idm \
_member_
```

CONFIGURING IDM INTEGRATION WITH TRIPLEO

For the initial deployment of a Red Hat OpenStack Platform environment, administrators can configure the TripleO deployment and management tool to include some of the configurations for IdM integration. These configurations include enabling domain support for LDAP back ends and defining LDAP back ends.

Enabling LDAP Domains

Set the **`KeystoneLDAPDomainEnable`** flag to **`true`** to enable LDAP domains in the TripleO environment file.

```
KeystoneLDAPDomainEnable: true
```

Enabling the **`KeystoneLDAPDomainEnable`** flag automatically sets the **`domain_specific_drivers_enabled`** option for the **`identity`** configuration group. By default, this configuration also configures the **`/etc/keystone/domains`** directory for the storage of domain configuration files.

Defining LDAP Back Ends

The **`KeystoneLDAPBackendConfigs`** parameter defines LDAP back-end configurations in TripleO heat templates. These definitions make use of various attributes to specify options for LDAP back-end configurations. The following example shows an environment file with an LDAP configuration matching the one shown previously for the identity domain, `idm`.

```
parameter_defaults:
  KeystoneLDAPDomainEnable: true
  KeystoneLDAPBackendConfigs:
    idm:
      url:  ldaps://idm.lab.example.com
      user: uid=ldaplookup,cn=users,cn=accounts,dc=lab,dc=example,dc=net
      user_filter:
 "(memberOf=cn=openstack,cn=groups,cn=accounts,dc=lab,dc=example,dc=net)"
      password: redhat
      user_tree_dn: cn=users,cn=accounts,dc=lab,dc=example,dc=net
      user_objectclass: inetUser
      user_id_attribute: uid
      user_name_attribute: uid
      user_mail_attribute: mail
      user_allow_create: false
      user_allow_update: false
      user_allow_delete: false
      group_objectclass = groupofnames
      group_tree_dn = cn=groups,cn=accounts,dc=lab,dc=example,dc=net
      group_id_attribute = cn
      group_name_attribute = cn
      group_allow_create = False
      group_allow_update = False
      group_allow_delete = False
```

```
use_tls = False
query_scope = sub
chase_referrals = false
tls_cacertfile: /etc/ssl/certs/ca.crt
```

REFERENCES

Domains
https://docs.openstack.org/security-guide/identity/domains.html

Domain-specific LDAP Back ends
https://docs.openstack.org/tripleo-docs/latest/install/advanced_deployment/domain_specific_ldap_backends.html

Further information is available in the *Identity Management Integration* chapter of the *Integrate with Identity Service Guide* for Red Hat OpenStack Platform 13; at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/integrate_with_identity_service/idm

Further information is available in the Installing and Uninstalling Identity Management Server chapter in the *Linux Domain Identity, Authentication, and Policy Guide* for Red Hat Enterprise Linux 7; at https://access.redhat.com/documentation/en-us/red_hat_enterprise_linux/7/html-single/linux_domain_identity_authentication_and_policy_guide/installing-ipa

VERIFYING AN IDM BACK-END CONFIGURATION

In this exercise, you will assign roles in OpenStack to users authenticated by Red Hat IdM.

OUTCOMES

You should be able to:

- Create projects in a domain.
- Assign roles to IdM users in a domain's project.
- Authenticate to dashboard using a Red Hat IdM back end.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab identity-idm setup`** command. This script ensures that the OpenStack services are running and that the environment is properly configured for this guided exercise.

```
[student@workstation ~]$ lab identity-idm setup
```

The classroom environment integrates the Red Hat OpenStack Platform (RHOSP) Identity Service with Red Hat IdM. It uses Red Hat IdM as an external authentication back end for the `Example` domain.

1. List the users in the `Example` domain to verify the integration with the IdM server. You should see a list of the `lab.example.net` realm users.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack user list --domain Example
+------------------------------------------------------------------+------------+
| ID                                                               | Name       |
+------------------------------------------------------------------+------------+
...output omitted...
| 8fe31f745b4f6dd29a431045d94dc24c3dd53f28a79c1fda1a11e981d57977d5 | architect2 |
...output omitted...
| b6cbbfb10d53eff2777a0235aeb2dd8de95fe4142ada32e2a803fd3b79c43029 | developer2 |
| 44c56514e3c2c86274b1bd581264b3b1349fcceb336f9457fa844d0f1c1ae225 | developer3 |
...output omitted...
+------------------------------------------------------------------+------------+
```

2. Create a project named `research` within the `Example` domain.

```
[student@workstation ~]$ openstack project create research --domain Example
+-------------+----------------------------------+
```

```
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 012f8a266990444da2130c6e8c68cdd1 |
| enabled     | True                             |
| id          | a6546caf46d746dfa5412ca0fc5c8c10 |
| is_domain   | False                            |
| name        | research                         |
| parent_id   | 012f8a266990444da2130c6e8c68cdd1 |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Assign the `_member_` role for the `research` project to the IdM `research-members` group.

```
[student@workstation ~]$ openstack role add \
--project research \
--project-domain Example \
--group research-members \
--group-domain Example \
_member_
```

4. Assign the `admin` role for the `research` project to the IdM `research-admins` group.

```
[student@workstation ~]$ openstack role add \
--project research \
--project-domain Example \
--group research-admins \
--group-domain Example \
admin
```

5. Log in to the dashboard to verify the role assigned to the `developer2` IdM user.
   5.1. On `workstation`, launch the Firefox web browser and connect to the dashboard at `http://dashboard.overcloud.example.com/`.
   5.2. Enter **`Example`** for the Domain, **`developer2`** for the User Name, and **`redhat`** for the Password to authenticate.
   5.3. After you have logged in to the dashboard, observe that the top navigation menu displays the Project and Identity tabs.
   5.4. Sign out of the dashboard.

6. Log in to the dashboard to verify the role assigned to the `developer3` IdM user.
   6.1. On `workstation`, launch the Firefox web browser and connect to the dashboard at `http://dashboard.overcloud.example.com/`.
   6.2. Enter **`Example`** for the Domain, **`developer3`** for the User Name, and **`redhat`** for the Password to authenticate.
   6.3. After you have logged in to the dashboard, observe that the top navigation menu displays the Project and Identity tabs.
   6.4. Sign out of the dashboard.

7. Log in to the dashboard to verify the role assigned to the `architect2` IdM user.
   7.1. On `workstation`, launch the Firefox web browser and connect to the dashboard at `http://dashboard.overcloud.example.com/`.
   7.2. Enter **`Example`** for the Domain, **`architect3`** for the User Name, and **`redhat`** for the Password to authenticate.
   7.3. After you have logged in to the dashboard, observe that the top navigation menu displays the Project, Admin, and Identity tabs, indicating that this user is assigned the `admin` role.
   7.4. Sign out of the dashboard.

Cleanup

On `workstation`, run the **`lab identity-idm cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab identity-idm cleanup
```

This concludes the guided exercise.

MANAGING IDENTITY SERVICE TOKENS

OBJECTIVES

After completing this section, students should be able to manage user tokens to implement user authorization for accessing OpenStack services.

AUTHENTICATION TOKENS

The Identity Service confirms a user's identity through an authentication process specified through plug-in configuration, and then provides that user with a token that represents the user's identity. A typical user token is *scoped*, meaning that it lists the resources and access for which it may be used. Tokens have a limited lifespan, allowing the user to perform service requests without further authentication until the token expires or is revoked. A scoped token lists the user's rights and privileges, as defined in roles relevant to the current project. When a user makes a request to an OpenStack service, the service verifies the requested resource access against the user's provided roles and then either allows or denies the requested service.

Any user may use the **`openstack token issue`** command to request a current scoped token with output showing the user id, the project, and the new token expiration. This token type is actually one of three types of authorization scope: unscoped, project-scoped, and domain-scoped. Because domains are a new feature supported in Identity Service v3, earlier documentation may refer only to scoped and unscoped tokens, in which the scope is project-based.

Authorization Scopes

TOKEN SCOPE	DESCRIPTION
Unscoped	Unscoped tokens are authentication-only tokens that do not contain a project, role, or service information payload. For example, an unscoped token may be used when authentication is provided by an identity provider other than the Identity Service, such as an LDAP, RADIUS, or AD server. The token is used to authenticate with the Identity Service, which then exchanges the unscoped token with the authenticated user's appropriate scoped token. An unscoped token may also be referred to as an Identity Service *default token*, which is not associated with a project or domain and may be exchanged for a scoped token.
Project-scoped	Project-scoped tokens provide authorization to perform operations on a service endpoint using the resources of a single project, allowing activities specified by the user's role in that project. These tokens contain the relevant service catalog, roles, and project information as payload and are considered to be associated to a specific project.
Domain-scoped	Domain-scoped tokens apply to services that occur at the domain level, rather than at the project or user level. This type of token has a payload containing the domain's service catalog, and is limited to services that do not require per-project endpoints. The token payload also contains the project and role information for the user within the specified domain.

TOKEN PROVIDERS

There are four types of token providers: UUID, PKI, PKIZ, and the newest provider, *Fernet* (pronounced fehr'nɛt). All tokens are comprised of a payload, in JSON or randomly generated UUID format, contained in a transport format, such as URL-friendly hexadecimal or cryptographic message syntax (CMS) packaging. The default recommended token provider for OpenStack has changed a few times, as OpenStack developers have attempted to address token size, security, and performance issues.

UUID Tokens

UUID tokens were the original and default token provider up until Red Hat OpenStack 2. They are 32 byte randomly generated UUIDs, which must be persistently stored in the Identity Service's configured back end to permit the Identity Service to validate the UUID each time a user makes a service request to any service endpoint. Although UUIDs are lightweight and easy to validate with a simple lookup, they have two disadvantages:

- First, because UUID tokens must be retained by the Identity Service back end for repetitive lookups, the storage space used grows as new tokens are generated. Until recently, expired tokens were not regularly purged from the back-end store, leading to service performance degradation over time.
- Second, every individual service API call must bundle the request and token together to send to the service component, where the service unpacks the UUID and sends a validation request to the Identity Service. The Identity Service looks up the token's identity to determine the roles and authorizations of the user, sending the information back to the resource service to determine if the service component will process the user request. This generates a tremendous amount of network traffic and activity to and from the Identity Service, which creates a scaling limitation.

PKI and PKIZ Tokens

Public Key Infrastructure (PKI) tokens were introduced in the Red Hat OpenStack 3 as a solution that would decrease the scale-limiting overhead on the Identity Service back-end and increase the security of tokens by using certificates and keys to sign and validate tokens. PKI uses a JSON payload, asymmetric keys, and the cryptographic message syntax (CMS) transport format. PKIZ tokens apply zlib compression after the JSON payload is signed to shrink the total token size, which typically exceeds 1600 bytes. The payload contains the service catalog with a size generally proportional to the number of service entries in the catalog.

The advantage of PKI tokens, because of the public key methodology, is the ability of the requested resource service component to verify and read the payload authorizations without needing to send the token back to the Identity Service for every request. To process request tokens, the requested service is only required to obtain the Identity Service's signing certificate, the current revocation list, and the CA public certificate that validates the signing certificate. Validated and unencoded tokens and payloads is stored and shared using Memcached, eliminating some repetitive token processing overhead.

The disadvantage of the PKI token provider method is unacceptable performance due to oversized shared caches, increased load on the Identity Service back end, and other problems associated with handling tokens with large payloads. PKI tokens take longer to create and to validate than UUID tokens. Consequently, UUID tokens replaced PKI and PKIZ tokens to once again became the recommended token provider. PKI and PKIZ token support was deprecated in the Mitaka release and was removed in the Ocata release.

Fernet Tokens

Fernet tokens are an implementation of a symmetric key cryptographic authentication method, which uses the same key to both encrypt and decrypt, designed specifically to process service API request tokens. Fernet supports using multiple keys, always using the first key (the current key) in the list to perform encryption, and then attempting earlier keys and *staged* keys (keys that

are scheduled to become current) to perform decryption. This technique allows Fernet keys to be rotated regularly for increased security, while still allowing tokens created with previous keys to be decrypted.

Fernet tokens do not exceed 250 bytes and are not persisted in the Identity Service back end. Fernet token payloads use the *MessagePack* binary serialization format to efficiently carry the authentication and authorization metadata, which is then encrypted and signed. Fernet tokens do not require persistence nor do they require maintenance, because they are created and validated instantaneously on any Identity Service node that can access the Fernet symmetric keys. The symmetric keys are stored and shared on all Identity Service nodes in a key repository located by default at **`/etc/keystone/fernet-keys/`**. The Fernet token provider was introduced in the Kilo release and became the default token provider in the Ocata release. In earlier OpenStack developer documentation, these tokens were referred to as *authenticated encryption (AE)* tokens.

> **WARNING**
> All of these token providers (UUID, PKI, PKIZ, and Fernet) are known as *bearer tokens*, which means that anyone holding the token can impersonate the user represented in that token without having to provide any authentication credentials. Bearer tokens must be protected from unnecessary disclosure to prevent unauthorized access.

TOKEN ADMINISTRATION

Token providers typically require minimal management tasks after they have been properly installed and configured. UUID tokens require the regular flushing of expired tokens. PKI token providers require maintenance of certificates, expiration, and revocations, as well as management of the persistent store. Fernet tokens require the regular rotation of keys for security, and distributing the key repository among all Identity Service nodes in a multinode HA deployment. Because Fernet token is now the default token provider, this section only discusses Fernet token administration tasks.

Fernet Keys

Fernet tokens do not require persistence but the Fernet symmetric keys must be accessible to all Identity Service nodes for validation of Fernet tokens. Because tokens should be replaced on a regular basis to minimize the ability to create impersonated Fernet tokens, the Fernet token provider uses a rotation method to put new symmetric keys into use without breaking the ability to decrypt Fernet tokens created with a previous key.

Fernet names its keys numerically based on each key's index in the rotation. It stores these keys in a key repository which by default is the **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys`** directory on the controller. Fernet uses three types of keys in its rotation process.

Primary Key
: The primary key is considered to be the current key. There can only be one primary key on a single Identity Service node, recognized because its file name always has the highest index number. Primary keys are used to both encrypt and decrypt Fernet tokens.

Secondary Key
: A secondary key is a key that was formerly a primary key and has been replaced (rotated out). It is only used to decrypt Fernet tokens; specifically, to decrypt any remaining Fernet tokens that it had originally encrypted. A secondary key's file is named with an index that is lower than the highest, but never has the index of **`0`**.

Staged Key

A staged key is a newly added key that will be the next primary key when the keys are next rotated. Similar to a secondary key, it is only used to decrypt tokens, which seems unnecessary because it has not yet been a primary key and has never encrypted tokens on the Identity Service node. However, in a multinode Identity Service configuration, after the key repository has been updated with a new staged key and distributed to all Identity Service nodes, those nodes will perform key rotation one at a time. A staged key on one node may be needed to decrypt tokens created by another node where that key has already become the primary key. The staged key is always recognized by having a file name with the index of **0**.

Rotating Fernet Keys

By default, the management of the overcloud's Fernet is performed by Director using Mistral. Administrators should use Mistral to initiate the rotation of Fernet keys. When initiated in this manner, Mistral stores the Fernet keys and these keys will persist across stack updates.

To initiate Fernet key rotation, execute the related Mistral workflow, `tripleo.fernet_keys.v1.rotate_fernet_keys`, as the `stack` user on Director.

```
[stack@director ~]$ source ~/stackrc
[stack@director ~]$ openstack workflow execution create
 tripleo.fernet_keys.v1.rotate_fernet_keys '{"container": "overcloud"}'
+-------------------+-----------------------------------------+
| Field             | Value                                   |
+-------------------+-----------------------------------------+
| ID                | 58c9c664-b966-4f82-b368-af5ed8de5b47    |
| Workflow ID       | 78f0990a-3d34-4bf2-a127-10c149bb275c    |
| Workflow name     | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Description       |                                         |
| Task Execution ID | <none>                                  |
| State             | RUNNING                                 |
| State info        | None                                    |
| Created at        | 2017-12-20 11:13:50                     |
| Updated at        | 2017-12-20 11:13:50                     |
+-------------------+-----------------------------------------+
```

Using the ID displayed in the workflow execution output, query the status of the workflow execution. When the workflow execution completes, the **`State`** of the workflow execution will display as **SUCCESS**.

```
[stack@director ~]$ openstack workflow execution show 58c9c664-b966-4f82-b368-
af5ed8de5b47
+-------------------+-----------------------------------------+
| Field             | Value                                   |
+-------------------+-----------------------------------------+
| ID                | 58c9c664-b966-4f82-b368-af5ed8de5b47    |
| Workflow ID       | 78f0990a-3d34-4bf2-a127-10c149bb275c    |
| Workflow name     | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Description       |                                         |
| Task Execution ID | <none>                                  |
| State             | SUCCESS                                 |
| State info        | None                                    |
| Created at        | 2017-12-20 11:13:50                     |
| Updated at        | 2017-12-20 11:15:00                     |
+-------------------+-----------------------------------------+
```

After the Fernet key rotation is complete, review the contents of the **/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys** directory on the controller. You will see a list of numbered files starting at **0**.

```
[heat-admin@controller ~]$ sudo ls /var/lib/config-data/puppet-generated/keystone/
etc/keystone/fernet-keys
0 1 2 3
```

The numbered file names indicate the index of the key contained in each file. The file named with the **0** index contains the new staged key. The file with the highest index as its file name contains the new primary key. All other files with names numbered in between the highest and lowest indices contain secondary keys.

Fernet Key Rotation Frequency

Regular rotation of Fernet keys limits the risk of attackers using compromised keys to gain unauthorized access to an OpenStack environment. However, if keys are rotated too frequently, it is possible to inadvertently remove secondary keys required for the decryption of valid tokens. To limit the removal of keys to those that are no longer used for token encryption and decryption, administrators need to correctly configure Fernet token parameters in the Identity service configuration file.

The first parameter to consider is **expiration**, which resides under the **token** section of the Identity service configuration file. This parameter dictates the lifespan of tokens in seconds. You can use the following command to display its value:

```
[heat-admin@controller ~]$ sudo crudini --get /var/lib/config-data/puppet-
generated/keystone/etc/keystone/keystone.conf token expiration
57600
```

The second parameter of importance is **max_active_keys**, which resides under the **fernet_tokens** section of the Identity service configuration file. This parameter dictates how many Fernet keys can be simultaneously active at any given time. You can use the following command to display its value:

```
[heat-admin@controller ~]$ sudo crudini --get /var/lib/config-data/puppet-
generated/keystone/etc/keystone/keystone.conf fernet_tokens max_active_keys
5
```

The interval of Fernet key rotation must not exceed the token **expiration** divided by two less than the value of **max_active_keys**. Therefore, in our example, with a token expiration of **57600** seconds and a maximum of **5** active Fernet keys, we should not rotate Fernet keys any more frequently than every 19,200 seconds, or 5 hours and 20 minutes. If more frequent rotation is desired, then either the token **expiration** parameter, the **max_active_keys** parameter, or both will need to be modified accordingly to ensure that there is no inadvertent removal of keys needed for the encryption or decryption of unexpired Fernet tokens.

REFERENCES

Keystone tokens

https://docs.openstack.org/keystone/latest/admin/identity-tokens.html

Further information is available in the *Using Fernet Tokens in the Overcloud* chapter of the *Deploy Fernet on the Overcloud Guide* for Red Hat OpenStack Platform 13; at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/deploy_fernet_on_the_overcloud/sec-fernet

MANAGING IDENTITY SERVICE TOKENS

In this exercise, you will view and rotate Fernet keys.

OUTCOMES

You should be able to:

- Initiate Fernet key rotation.
- Validate Fernet key rotation.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab identity-tokens setup`** command. This script ensures that the OpenStack services are running and that the environment is properly configured for this guided exercise.

```
[student@workstation ~]$ lab identity-tokens setup
```

1. Use the **`crudini`** command to determine whether or not the overcloud uses Fernet tokens.

   1.1. Log in to the `controller0` system then become the `root` user.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

   1.2. List the value of the **`provider`** parameter in the **`token`** section of the **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf`** Identity service configuration file to determine the token provider.

```
[root@controller0 ~]# crudini --get \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
token provider
fernet
```

   1.3. List the value of the **`driver`** parameter in the **`token`** section of the **`/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf`** Identity service configuration file to determine the token driver.

```
[root@controller0 ~]# crudini --get \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
token driver
sql
```

2. Use the **crudini** command to determine the location of the Fernet key repository, and the maximum number of active Fernet keys.

   2.1. List the value of the **key_repository** parameter in the **fernet_tokens** section of the **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** Identity service configuration file to determine the location of the Fernet key repository.

```
[root@controller0 ~]# crudini --get \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
fernet_tokens key_repository
/etc/keystone/fernet-keys
```

   2.2. List the value of the **max_active_keys** parameter in the **fernet_tokens** section of the **/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf** Identity service configuration file to determine the maximum number of active Fernet keys.

```
[root@controller0 ~]# crudini --get \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/keystone.conf \
fernet_tokens max_active_keys
5
```

3. Display the list of active Fernet keys.

```
[root@controller0 ~]# ls -l \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/
total 8
-rw-------. 1 42425 42425 44 Aug  1 15:38 0
-rw-------. 1 42425 42425 44 Aug  1 15:38 1
```

4. Display the contents of the active Fernet key files. The file is not terminated by a line feed so use the **cat** command in conjunction with the **echo** command to make it easier to view the contents of the key files. Note that the output might be slightly different on your system.

```
[root@controller0 ~]# cat \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/0; echo
dXsGBGMEbbPutvd-tASd77aqVFox26TFBPsArA3Rpb8=
[root@controller0 ~]# cat \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/1; echo
HyOrU0K_W0u7BWPTQ_rb0FLd4-ojilko_x27Cpv28zw=
```

5. Rotate the Fernet keys.

   5.1. In a new terminal session, log in to the `director` system.

```
[student@workstation ~]$ ssh director
(undercloud) [stack@director ~]$
```

   5.2. Execute the Mistral workflow to initiate the rotation of the Fernet keys.

```
(undercloud) [stack@director ~]$ openstack workflow execution create \
tripleo.fernet_keys.v1.rotate_fernet_keys '{"container": "overcloud"}'
```

```
+--------------------+------------------------------------------+
| Field              | Value                                    |
+--------------------+------------------------------------------+
| ID                 | d2fe1059-c3f0-40e3-b31a-eb83bf383711     |
| Workflow ID        | 42e56b9a-b174-4626-831d-0e66b2fe0f32     |
| Workflow name      | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Workflow namespace |                                          |
| Description        |                                          |
| Task Execution ID  | <none>                                   |
| State              | RUNNING                                  |
| State info         | None                                     |
| Created at         | 2018-08-14 01:51:33                      |
| Updated at         | 2018-08-14 01:51:33                      |
+--------------------+------------------------------------------+
```

5.3. Wait two minutes or so and then view the status of the workflow execution. Use the workflow execution ID displayed in the output of the previous command. Continue when the state displays **SUCCESS**.

```
(undercloud) [stack@director ~]$ openstack workflow execution show \
d2fe1059-c3f0-40e3-b31a-eb83bf383711
+--------------------+------------------------------------------+
| Field              | Value                                    |
+--------------------+------------------------------------------+
| ID                 | d2fe1059-c3f0-40e3-b31a-eb83bf383711     |
| Workflow ID        | 42e56b9a-b174-4626-831d-0e66b2fe0f32     |
| Workflow name      | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Workflow namespace |                                          |
| Description        |                                          |
| Task Execution ID  | <none>                                   |
| State              | SUCCESS                                  |
| State info         | None                                     |
| Created at         | 2018-08-14 01:51:33                      |
| Updated at         | 2018-08-14 01:52:56                      |
+--------------------+------------------------------------------+
```

6. Return to the `controller0` system and list the active Fernet keys in the repository directory.

```
[root@controller0 ~]# ls -l \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/
total 8
-rw-------. 1 42425 42425 44 Aug  1 15:38 0
-rw-------. 1 42425 42425 44 Aug  1 15:38 1
-rw-------. 1 42425 42425 44 Aug  1 15:38 2
```

7. Display the contents of the active Fernet key files.

```
[root@controller0 ~]# cat \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/0; echo
V8oBbRwmgwft2kJmPelsYFUX3Qbym15RZwNQEl6lrOk=
[root@controller0 ~]# cat \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/1; echo
HyOrU0K_W0u7BWPTQ_rb0FLd4-ojilko_x27Cpv28zw=
```

```
[root@controller0 ~]# cat \
/var/lib/config-data/puppet-generated/keystone/etc/keystone/fernet-keys/2; echo
dXsGBGMEbbPutvd-tASd77aqVFox26TFBPsArA3Rpb8=
```

8. Notice that the contents of the previous staged key file, **0**, has now rotated into the highest indexed key file, **2**, and is now the primary key. Notice also that the contents of the previous primary key file, **1**, is now in the second highest indexed key file, **1**, and is now the secondary key. Lastly, notice that the staged key file, **0**, now contains a new staged key.

Cleanup

On `workstation`, run the **`lab identity-tokens cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab identity-tokens cleanup
```

This concludes the guided exercise.

MANAGING PROJECT ORGANIZATION

OBJECTIVES

After completing this section, students should be able to manage project quotas, domains, hierarchies, and groups.

DOMAINS

New to Identity Service v3, domains serve as containers for users, groups, and projects. The Red Hat OpenStack Platform initially installs a single domain named `Default`. To create additional domains, use the **`openstack domain create`** command.

```
[user@demo ~]$ openstack domain create DOMAIN
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| enabled     | True                             |
| id          | 06021b952aba445e89c6ecfbbf274b8a |
| name        | DOMAIN                           |
| tags        | []                               |
+-------------+----------------------------------+
```

Use the **`openstack domain list`** command to display the list of domains.

```
[user@demo ~]$ openstack domain list
+----------------------------------+------------+---------+--------------------+
| ID                               | Name       | Enabled | Description        |
+----------------------------------+------------+---------+--------------------+
| 06021b952aba445e89c6ecfbbf274b8a | consulting | True    |                    |
| 7721873168994ff9baacca97772b5856 | heat_stack | True    |                    |
| ca08e25f26f144c58ad61af181ee3450 | Example    | True    |                    |
| default                          | Default    | True    | The default domain |
+----------------------------------+------------+---------+--------------------+
```

Users

Administrators should reserve the `Default` domain for use with service accounts. User accounts, on the other hand, should reside in other domains. Administrators can create new domains and then populate them locally with users.

To add a new user to a domain, use the **`openstack user create`** command and specify the domain with the **`--domain`** option. Without the **`--domain`** option, the command creates the user in the `Default` domain. The **`--password-prompt`** option allows the administrator to interactively type in the password at the prompt. Specifying the password in the command line is not recommended as it leaves a risk of exploitation of the command history to compromise user's password.

```
[user@demo ~]$ openstack user create USER --domain DOMAIN --password-prompt
User Password: PASSWORD
```

```
Repeat User Password: PASSWORD
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | e2bd1f413aa7451db3d3fe362bda4514 |
| enabled             | True                             |
| id                  | 3b3fe754267b42e4b0d9f9d342dac5bc |
| name                | USER                             |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

Domains also introduce the possibility for integration with an authentication back end, such as LDAP or Active Directory. With this integration, Identity Service v3 leverages existing user accounts in an identity management back end.

Groups

Identity Service v3 also introduces the concept of *groups*. The group entity facilitates bulk management of user access to resources. Rather than granting access on a per user basis, groups allow administrators to perform a single administrative operation on a group and have its effects distributed to all members of the group.

To create a new group, use the **`openstack group create`** command and specify the domain with the **`--domain`** option. Without the **`--domain`** option, the command creates the group in the `Default` domain.

```
[user@demo ~]$ openstack group create GROUP --domain DOMAIN
+-------------+----------------------------------+
| description |                                  |
| domain_id   | e2bd1f413aa7451db3d3fe362bda4514 |
| id          | a0a39b7aef58484d8fe1294df0a1d5a5 |
| name        | GROUP                            |
+-------------+----------------------------------+
```

To add a user to a group, use the **`openstack group add user`** command. You can add a user to any group, not just to groups in the same domain as the user. Specify the user domain with the **`--user-domain`** option, and specify the group domain with the **`--group-domain`** option.

```
[user@demo ~]$ openstack group add user --group-domain GROUPDOMAIN \
--user-domain USERDOMAIN GROUP USER
```

For domains integrated with external authentication back ends, Identity Service v3 can also leverage existing user groups in the identity management back end. By querying the back end, Identity Service v3 can determine the members of each group.

NOTE

Even though you can add users of one domain to groups in other domains, this cross-domain operation is only allowed for domains with shared Identity Service back ends. If the domains use different back ends, you will see the following error:

```
[user@demo ~]$ openstack group add user --group-domain GROUPDOMAIN \
--user-domain USERDOMAIN GROUP USER
USER not added to group GROUP: Group membership across backend boundaries
 is not allowed. Group in question is 573353d9ae674ef79d477ecb8c66f946, user
 is USER. (HTTP 403) (Request-ID: req-90c9113e-7239-4c7f-bc79-33444e973bba)
1 of 1 users not added to group GROUP.
```

Projects

In previous releases of Red Hat OpenStack Platform, there was a one-to-one relationship between users and projects. This meant that users could not belong to more than one project. With the introduction of domains in Identity Service v3, users are no longer contained by projects but rather by domains. With this design change, users can now belong to more than one project.

To create a project within a domain, use the **`openstack project create`** command. Use the **`--domain`** option to specify the domain within which to create the new project. Without the **`--domain`** option, the command creates the project in the `Default` domain.

```
[user@demo ~]$ openstack project create PROJECT --domain DOMAIN
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | e2bd1f413aa7451db3d3fe362bda4514 |
| enabled     | True                             |
| id          | 52d9fbf0883d427a87880be38fd929f5 |
| is_domain   | False                            |
| name        | PROJECT                          |
| parent_id   | e2bd1f413aa7451db3d3fe362bda4514 |
| tags        | []                               |
+-------------+----------------------------------+
```

HIERARCHICAL MULTITENANCY

Prior to the introduction of domains, projects existed in a flat structure. This did not allow administrators to easily represent their organization's hierarchy. For example, organizations are often structured in a hierarchy consisting of departments, which may contain subdepartments, teams, or other smaller groups.

Although domains allow a departure from a single, flat collection of projects, this only allows for a two-level hierarchy. Identity Service v3 allows for a multilevel hierarchy by introducing the ability to nest projects.

Nested Projects

Nested projects must be members of the same domain. To create a nested project, use the **`openstack project create`** command. Use the **`--domain`** option to specify the domain in

which to create the new project. Use the **--parent** option to specify the parent project for the new project.

```
[user@demo ~]$ openstack project create SUBPROJECT --parent PARENTPROJECT \
--domain DOMAIN
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | default                          |
| enabled     | True                             |
| id          | 1ae7dc044ff94e88b38d9729b29657ff |
| is_domain   | False                            |
| name        | SUBPROJECT                       |
| parent_id   | 27868b46974a4ffe84b790fb94f1acde |
| tags        | []                               |
+-------------+----------------------------------+
```

Nested Quotas

Quotas offer administrators the ability to manage system resource utilization in a Red Hat OpenStack Platform environment. Administrators can enforce quotas at a project level and at a project-user level.

With the advent of nested projects in a hierarchical multitenant design, project quotas also need to be structured hierarchically. Quotas for subprojects should be allocated from the quota assigned to their parent projects. This concept is known as *nested quotas*.

Unfortunately, nested quotas are not supported in the current version of Red Hat OpenStack Platform. Quotas assigned to projects and subprojects have no relationship to each other. This creates a situation where it is possible for subprojects to allocate more resources than allowed by the quotas applied to their parent projects.

> **WARNING**
> Until Red Hat OpenStack Platform implements nested quotas, administrators need to manually manage the quotas of projects and their subprojects to prevent excessive allocation and consumption of resources.

ROLE MANAGEMENT

In a Red Hat OpenStack Platform environment, users must authenticate using a project scope. Therefore, each user must be granted access to at least one project. Identity Service makes use of Role-based Access Control (RBAC). Administrators can grant access to domains and projects by assigning roles to users or groups. A user or group can be assigned roles for any domain and project, even projects outside the domain that contains the user or group.

To grant a role, use the **openstack role add** command. When granting a role to a user, use the **--user** option to specify the user to grant a role to and the **--user-domain** option to specify the domain the user belongs to. When granting a role to a group, use the **--group** option to specify the group to grant a role to and the **--group-domain** option to specify the domain the group belongs to.

To grant role access for a domain, use the **--domain** option to specify the domain. To grant role access for a project, use the **--project** option to specify the project and **--project-domain** to specify the domain the project resides in.

The following example demonstrates granting the `admin` user of the `Default` domain the `admin` role for the `Example` domain.

```
[user@demo ~]$ openstack role add --domain Example --user admin admin
```

The following example demonstrates granting the `developers` group of the `Lab` domain the `_member_` role for the `research` project in the `Example` domain.

```
[user@demo ~]$ openstack role add --project research --project-domain Example \
--group developers --group-domain Lab _member_
```

Inherited Roles

Prior to the introduction of nested projects in Identity Service v3, administrators had to assign roles for each project individually. To facilitate the management of roles in the new hierarchical multitenant environment, Identity Service v3 also introduces the concept of role inheritance. The **`openstack role add`** command now includes the **`--inherited`** option. This options allows the role assignment to be automatically inherited by all current and future subprojects of a project.

The following example demonstrates granting the `developers` group of the `Lab` domain the `_member_` role to all current and future subprojects of the `research` project in the `Example` domain.

```
[user@demo ~]$ openstack role add --project research --project-domain Example \
--group developers --group-domain Lab --inherited _member_
```

Viewing Roles

To view the role assignments for a domain or project, use the **`openstack role assignment list`** command. When viewing the role assignment for a domain, use the **`--domain`** option to specify the domain. Use the **`--project`** and **`--project-domain`** options to specify the project and project domain for which to display the role assignment.

The following examples demonstrate assigning the `research-members` group in the `Example` domain the `_member_` role for the `test` project in the `Example` domain as a normal role and in the `true` project in the `Example` domain as an inherited role.

```
[user@demo ~]$ openstack role add --group research-members --group-domain \
Example --project test --project-domain Example _member_
[user@demo ~]$ openstack role add --group research-members --group-domain \
Example --project true --project-domain Example --inherited _member_
```

WARNING

Assigning a role to a project using the **`--inherited`** option with the **`openstack role add`** command only assigns the role to the current and future subprojects of the project. The added role only applies to the subprojects and not to the project specified in the command.

To view the roles assigned to the `test` project in the `Example` domain, use the following command. The **`Inherited`** column indicates whether a role is inherited or not.

```
[user@demo ~]$ openstack role assignment list \
```

```
--project test --project-domain Example
+---------+------+---------+---------+--------+-----------+
| Role    | User | Group   | Project | Domain | Inherited |
+---------+------+---------+---------+--------+-----------+
| 9f...ab |      | c1...7e | 1e...05 |        | False     |
| 9f...ab |      | c1...7e | 1e...05 |        | True      |
+---------+------+---------+---------+--------+-----------+
```

Use the **--names** option to view the role assignment as names rather than IDs.

```
[user@demo ~]$ openstack role assignment list \
--project test --project-domain Example --names
+----------+------+-------------------------+--------------+--------+-----------+
| Role     | User | Group                   | Project      | Domain | Inherited |
+----------+------+-------------------------+--------------+--------+-----------+
| _member_ |      | research-members@Example | test@Example |        | False     |
| _member_ |      | research-members@Example | true@Example |        | True      |
+----------+------+-------------------------+--------------+--------+-----------+
```

Identity Service v3 includes the **--inherited** option to only view role assignments inherited by subprojects.

```
[user@demo ~]$ openstack role assignment list \
--project test --project-domain Example --names --inherited
+----------+------+-------------------------+--------------+--------+-----------+
| Role     | User | Group                   | Project      | Domain | Inherited |
+----------+------+-------------------------+--------------+--------+-----------+
| _member_ |      | research-members@Example | true@Example |        | True      |
+----------+------+-------------------------+--------------+--------+-----------+
```

Identity Service v3 also includes the **--effective** option to view role assignments in terms of users even for roles assigned to groups.

```
[user@demo ~]$ openstack role assignment list \
--project test --project-domain Example --names --effective
+----------+--------------------+-------+--------------+--------+-----------+
| Role     | User               | Group | Project      | Domain | Inherited |
+----------+--------------------+-------+--------------+--------+-----------+
| _member_ | developer1@Example |       | test@Example |        | False     |
| _member_ | developer2@Example |       | test@Example |        | False     |
| _member_ | developer3@Example |       | test@Example |        | False     |
...output omitted...
| _member_ | architect7@Example |       | test@Example |        | False     |
| _member_ | architect8@Example |       | test@Example |        | False     |
| _member_ | architect9@Example |       | test@Example |        | False     |
+----------+--------------------+-------+--------------+--------+-----------+
```

A few other useful options included in Identity Service v3 are the **--user**, **--user-domain**, **--group**, **--group-domain** options for filtering the list of role assignments by either user or group. The following example shows how the results of the previous role assignment query can be filtered for the `developer1` user in the `Example` domain.

```
[user@demo ~]$ openstack role assignment list \
```

```
--project test --project-domain Example --names \
--effective --user developer1 --user-domain Example
+----------+--------------------+-------+--------------+--------+-----------+
| Role     | User               | Group | Project      | Domain | Inherited |
+----------+--------------------+-------+--------------+--------+-----------+
| _member_ | developer1@Example |       | test@Example |        | False     |
+----------+--------------------+-------+--------------+--------+-----------+
```

REFERENCES

Further information is available in the *Users and Identity Management Guide* for Red Hat Enterprise OpenStack Platform 13; at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/html-single/users_and_identity_management_guide/

MANAGING PROJECT ORGANIZATION

In this exercise, you will create nested projects within a domain.

OUTCOMES

You should be able to:

- Create domains, projects, and subprojects.
- Create groups and assign group roles.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab identity-project setup`** command. This script ensures that the OpenStack services are running and that the environment is properly configured for this guided exercise.

```
[student@workstation ~]$ lab identity-project setup
```

1. On `workstation`, source the **`~/admin-rc`** file. Create a new domain named `Lab`.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack domain create Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| enabled     | True                             |
| id          | 8f0c26484c444e3687858c47780663d2 |
| name        | Lab                              |
| tags        | []                               |
+-------------+----------------------------------+
```

2. Create a new project named `support`, within the `Lab` domain.

```
[student@workstation ~]$ openstack project create support --domain Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 8f0c26484c444e3687858c47780663d2 |
| enabled     | True                             |
| id          | 46b949d1c59740f68f11e552fd2635b7 |
| is_domain   | False                            |
| name        | support                          |
| parent_id   | 8f0c26484c444e3687858c47780663d2 |
```

```
| tags         | []                                |
+-------------+----------------------------------+
```

3. Create a new project named `dev` within the `support` project.

```
[student@workstation ~]$ openstack project create dev --parent support \
--domain Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 8f0c26484c444e3687858c47780663d2 |
| enabled     | True                             |
| id          | ba9aa83cb4a04075a16bd976545f8176 |
| is_domain   | False                            |
| name        | dev                              |
| parent_id   | 46b949d1c59740f68f11e552fd2635b7 |
| tags        | []                               |
+-------------+----------------------------------+
```

4. Create a new user group named `developers` within the `Lab` domain.

```
[student@workstation ~]$ openstack group create developers --domain Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 8f0c26484c444e3687858c47780663d2 |
| id          | 8eadf3afdda44049acbb03229bff2fa1 |
| name        | developers                       |
+-------------+----------------------------------+
```

5. Assign the `developers` group in the `Lab` domain the `_member_` role for the `support` project in the `Lab` domain.

```
[student@workstation ~]$ openstack role add --group developers \
--group-domain Lab --project support --project-domain Lab _member_
```

6. Assign the `developers` group in the `Lab` domain the `_member_` role for all the subprojects of the `support` project in the `Lab` domain. Use the **`--inherited`** flag to have the role inherited by subprojects of the `support` project.

```
[student@workstation ~]$ openstack role add --group developers \
--group-domain Lab --project support --project-domain Lab --inherited _member_
```

7. Create a new user, `developer4`, in the `Lab` domain. Set the password for the new user to `redhat`.

```
[student@workstation ~]$ openstack user create developer4 --domain Lab \
--password redhat
+---------------------+----------------------------------+
```

```
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | 8f0c26484c444e3687858c47780663d2 |
| enabled             | True                             |
| id                  | 94ad108ffe49473bbef850123496a324 |
| name                | developer4                       |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

8. Add the `developer4` user of the `Lab` domain to the `developers` user group of the `Lab` domain.

```
[student@workstation ~]$ openstack group add user --group-domain Lab \
--user-domain Lab developers developer4
```

9. Verify the role assignments for the `support` project in the `Lab` domain.

```
[student@workstation ~]$ openstack role assignment list --project support \
--project-domain Lab --names --effective
+----------+-----------------+-------+--------------+--------+-----------+
| Role     | User            | Group | Project      | Domain | Inherited |
+----------+-----------------+-------+--------------+--------+-----------+
| _member_ | developer4@Lab  |       | support@Lab  |        | False     |
+----------+-----------------+-------+--------------+--------+-----------+
```

10. Verify the role assignments for the `dev` subproject in the `Lab` domain.

```
[student@workstation ~]$ openstack role assignment list --project dev \
--project-domain Lab --names --effective
+----------+-----------------+-------+----------+--------+-----------+
| Role     | User            | Group | Project  | Domain | Inherited |
+----------+-----------------+-------+----------+--------+-----------+
| _member_ | developer4@Lab  |       | dev@Lab  |        | True      |
+----------+-----------------+-------+----------+--------+-----------+
```

Cleanup

On `workstation`, run the **`lab identity-project cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab identity-project cleanup
```

This concludes the guided exercise.

INTEGRATING IDENTITY MANAGEMENT

PERFORMANCE CHECKLIST

In this lab, you will create nested projects within a domain and assign roles in OpenStack to users authenticated by Red Hat IdM. You will also initiate and validate a Fernet key rotation.

OUTCOMES

You should be able to:

- Create projects and subprojects in a domain.
- Assign group roles to a group in a domain.
- Identify roles assigned to users in a domain.
- Initiate and validate Fernet key rotation.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab identity-review setup`** command. This ensures that the OpenStack services are running and that the environment has been properly configured for this lab.

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

1. Create a project called `consulting` in the `Example` domain.
2. Create a new subproject named `development` in the `consulting` project.
3. Assign the `consulting-members` group in the `Example` domain the `_member_` role to the `consulting` project in the `Example` domain.
4. Assign the `consulting-members` group in the `Example` domain the `admin` role for all the subprojects of the `consulting` project in the `Example` domain.
5. Verify the roles assigned to the following users in the `consulting` project within the `Example` domain:
   - `operator1`
   - `architect1`
6. Verify the roles assigned to the following users in the `development` subprojects in the `Example` domain.
   - `operator1`
   - `architect1`
7. On `director`, initiate the rotation of the overcloud Fernet keys. Verify the completion of Fernet keys rotation.

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab identity-review cleanup
```

This concludes the lab.

INTEGRATING IDENTITY MANAGEMENT

PERFORMANCE CHECKLIST

In this lab, you will create nested projects within a domain and assign roles in OpenStack to users authenticated by Red Hat IdM. You will also initiate and validate a Fernet key rotation.

OUTCOMES

You should be able to:

- Create projects and subprojects in a domain.
- Assign group roles to a group in a domain.
- Identify roles assigned to users in a domain.
- Initiate and validate Fernet key rotation.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab identity-review setup`** command. This ensures that the OpenStack services are running and that the environment has been properly configured for this lab.

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

1. Create a project called `consulting` in the `Example` domain.

   1.1. Source the admin environment file, **`/home/student/admin-rc`**. Create the `consulting` project in the `Example` domain.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack project create consulting --domain Example
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 0ed0e18ad28747ccac32ec344e271d58 |
| enabled     | True                             |
| id          | 191dd8a9c767498ca21587c48f01d96c |
| is_domain   | False                            |
| name        | consulting                       |
| parent_id   | 0ed0e18ad28747ccac32ec344e271d58 |
| tags        | []                               |
+-------------+----------------------------------+
```

   1.2. Verify that the `consulting` project is created in the `Example` domain.

```
[student@workstation ~]$ openstack project list --domain Example
+----------------------------------+------------+
| ID                               | Name       |
+----------------------------------+------------+
| 191dd8a9c767498ca21587c48f01d96c | consulting |
+----------------------------------+------------+
```

2. Create a new subproject named `development` in the `consulting` project.

```
[student@workstation ~]$ openstack project create development \
--parent consulting --domain Example
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | 0ed0e18ad28747ccac32ec344e271d58 |
| enabled     | True                             |
| id          | 547e84f2efeb431ea77c48aed5a14a68 |
| is_domain   | False                            |
| name        | development                      |
| parent_id   | 191dd8a9c767498ca21587c48f01d96c |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Assign the `consulting-members` group in the `Example` domain the `_member_` role to the `consulting` project in the `Example` domain.

```
[student@workstation ~]$ openstack role add \
--group consulting-members --group-domain Example \
--project consulting --project-domain Example _member_
```

4. Assign the `consulting-members` group in the `Example` domain the `admin` role for all the subprojects of the `consulting` project in the `Example` domain.

```
[student@workstation ~]$ openstack role add \
--group consulting-members --group-domain Example \
--project consulting --project-domain Example --inherited admin
```

5. Verify the roles assigned to the following users in the `consulting` project within the `Example` domain:

   - `operator1`
   - `architect1`

   5.1. Verify that the `_member_` role is assigned to the `operator1` user in the `consulting` project in the `Example` domain.

```
[student@workstation ~]$ openstack role assignment list \
--user operator1 --user-domain Example \
--project consulting --names --effective
+----------+------------------+-----+--------------------+-----+-----------+
| Role     | User             | ... | Project            | ... | Inherited |
+----------+------------------+-----+--------------------+-----+-----------+
| _member_ | operator1@Example |     | consulting@Example |     | False     |
```

```
+----------+-------------------+-----+-------------------+-----+-----------+
```

5.2. Verify that the `_member_` role is assigned to the `architect1` user in the `consulting` project in the `Example` domain.

```
[student@workstation ~]$ openstack role assignment list \
--user architect1 --user-domain Example \
--project consulting --names --effective
+----------+-------------------+-----+-------------------+-----+-----------+
| Role     | User              | ... | Project           | ... | Inherited |
+----------+-------------------+-----+-------------------+-----+-----------+
| _member_ | architect1@Example |     | consulting@Example |     | False     |
+----------+-------------------+-----+-------------------+-----+-----------+
```

6. Verify the roles assigned to the following users in the `development` subproject in the `Example` domain.

- `operator1`
- `architect1`

6.1. Verify that the `admin` role is assigned to the `operator1` user in the `development` subproject in the `Example` domain.

```
[student@workstation ~]$ openstack role assignment list \
--user operator1 --user-domain Example \
--project development --names --effective
+-------+-------------------+-----+--------------------+-----+-----------+
| Role  | User              | ... | Project            | ... | Inherited |
+-------+-------------------+-----+--------------------+-----+-----------+
| admin | operator1@Example |     | development@Example |     | True      |
+-------+-------------------+-----+--------------------+-----+-----------+
```

6.2. Verify that the `admin` role is assigned to the `architect1` user in the `development` subproject in the `Example` domain.

```
[student@workstation ~]$ openstack role assignment list \
--user architect1 --user-domain Example \
--project development --names --effective
+-------+-------------------+-----+--------------------+-----+-----------+
| Role  | User              | ... | Project            | ... | Inherited |
+-------+-------------------+-----+--------------------+-----+-----------+
| admin | architect1@Example |     | development@Example |     | True      |
+-------+-------------------+-----+--------------------+-----+-----------+
```

7. On `director`, initiate the rotation of the overcloud Fernet keys. Verify the completion of Fernet keys rotation.

7.1. Log in to the `director` system.

```
[student@workstation ~]$ ssh director
(undercloud) [stack@director ~]$
```

7.2. Execute the Mistral workflow to initiate the rotation of the Fernet keys.

```
(undercloud) [stack@director ~]$ openstack workflow execution create \
tripleo.fernet_keys.v1.rotate_fernet_keys '{"container": "overcloud"}'
```

```
+--------------------+------------------------------------------+
| Field              | Value                                    |
+--------------------+------------------------------------------+
| ID                 | b41b09b9-f9ad-4924-9b6d-b9b11bd613f6     |
| Workflow ID        | 01bd8112-405d-4f0d-9aea-83c642d82946     |
| Workflow name      | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Workflow namespace |                                          |
| Description        |                                          |
| Task Execution ID  | <none>                                   |
| State              | RUNNING                                  |
| State info         | None                                     |
| Created at         | 2018-09-06 11:49:41                      |
| Updated at         | 2018-09-06 11:49:41                      |
+--------------------+------------------------------------------+
```

7.3. Verify the status of the workflow execution by referencing its ID displayed in the output of the previous command. Rerun the command if required, and wait until the state is set to **SUCCESS**.

```
(undercloud) [stack@director ~]$ openstack workflow execution show \
b41b09b9-f9ad-4924-9b6d-b9b11bd613f6
+--------------------+------------------------------------------+
| Field              | Value                                    |
+--------------------+------------------------------------------+
| ID                 | b41b09b9-f9ad-4924-9b6d-b9b11bd613f6     |
| Workflow ID        | 01bd8112-405d-4f0d-9aea-83c642d82946     |
| Workflow name      | tripleo.fernet_keys.v1.rotate_fernet_keys |
| Workflow namespace |                                          |
| Description        |                                          |
| Task Execution ID  | <none>                                   |
| State              | SUCCESS                                  |
| State info         | None                                     |
| Created at         | 2018-09-06 11:49:41                      |
| Updated at         | 2018-09-06 11:50:44                      |
+--------------------+------------------------------------------+
```

Log off from `director`.

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab identity-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- Identity Service can be configured to authenticate users from external IdM environments.
- To perform user credential authentication using a Red Hat IdM back end, the identity service requires an LDAP look-up account on the IdM server.
- There are four types of token providers: UUID, PKI, PKIZ, and Fernet. The Fernet tokens are enabled by default since Red Hat OpenStack Platform 12.
- Fernet tokens have a maximum limit of 250 bytes, which makes them small enough to be ideal for API calls and minimize the data kept on disk. Each Fernet token is actually composed of two smaller keys: a 128-bit AES encryption key and a 128-bit SHA256 HMAC signing key.
- Domains provide granularity in the authorization model. With domains, the resource mapping can be summed up as: a domain is made up of users and projects, wherein users can have roles at the project and domain level.

CHAPTER 4

PERFORMING IMAGE OPERATIONS

GOAL

Customize instance deployments by creating build-to-order images, modifying guest and disk images, and initializing instances when deployed.

OBJECTIVES

- Describe common image formats, features, and use cases.
- Build an image using **diskimage-builder**.
- Modify existing images with additional software and configuration using **`guestfish`** and **`virt-customize`**.
- Customize instances during deployment using **`cloud-init`**.

SECTIONS

- Comparing Image Formats (and Quiz)
- Building Customized Images (and Guided Exercise)
- Modifying Guest and Disk Images (and Guided Exercise)
- Initializing an Instance During Deployment (and Guided Exercise)

LAB

- Performing Image Operations

COMPARING IMAGE FORMATS

OBJECTIVE

After completing this section, students should be able to describe the common image formats used within Red Hat OpenStack Platform.

COMMON IMAGE FORMATS

Building custom images is one way to standardize software configuration and features across your organization. Image customization might include software packages, security hardening, third-party monitoring or backup agents, and operator or automation accounts with SSH keys.

Red Hat OpenStack Platform, using the Linux KVM hypervisor and QEMU emulator technologies, supports many virtual disk image formats, either directly or through QEMU conversion. Red Hat OpenStack Platform primarily uses RAW and QCOW2 formats, which are explained in this section.

Image Format Overview

FORMAT	DESCRIPTION
RAW	An unstructured disk image format that is an exact copy of the original disk. These images may have an extension such as `.img`, `.raw` or `.bin`.
QCOW2	The Qemu Copy On Write v2 format. This format is supported by the QEMU emulator and can expand space dynamically. This is the most common format used with Linux KVM.
ISO	An archive format for the data contents of a CD or DVD optical disc.
AKI	An Amazon kernel image. Not typically used with Linux KVM or Red Hat OpenStack Platform.
AMI	An Amazon machine image. Not typically used with Linux KVM or Red Hat OpenStack Platform.
ARI	An Amazon ramdisk image. Not typically used with Linux KVM or Red Hat OpenStack Platform.
VDI	A Virtualbox disk image, originally for the VirtualBox hosted hypervisor. Supported by the QEMU emulator.
VHD	The Virtual Hard Disk format, used by Microsoft's Hyper-V and originally by Windows Virtual PC. Also supported by VMware, Xen, Virtualbox and others, but not a universally popular format.
VMDK	The virtual machine disk format, originally created for VMware, but now a common, open format that is universally supported.

The RAW format is a bootable, uncompressed virtual disk image. No formatting is applied to disk images stored in RAW format. Data operations on RAW disk images require no additional format handling. Data written to a disk location offset is written to the same offset in the backing file or volume. This is the basis for increased RAW image performance.

The QCOW2 format is a space-optimized file that minimizes network bandwidth and volume operation sizes. Although QCOW2 use requires real-time handling, the format supports popular features such as snapshots, copy-on-write, and encryption, which are also dynamic.

Comparison of RAW and QCOW2 Image Formats

ATTRIBUTE	RAW	QCOW2
Image Size	A RAW image is a bit-equivalent copy of the source disk, including empty space. In file systems or back ends that support sparse storage, RAW images typically use less actual space than they report.	QCOW2 is a space-optimized representation of the virtual disk image. Not to be confused with compression or sparse storage, the optimization is internal to the file structure. Commonly, a QCOW2 file is smaller than a RAW image from the same source. QCOW2 also supports zlib compression. QCOW2 files are already optimized and consequently might not use sparse file system or storage features.
Performance	RAW format performance is higher than QCOW2 performance because disk space is preallocated at instance deployment, avoiding latencies associated with on-demand space allocation. RAW images do not require additional processing during use.	QCOW2 format performance is lower than RAW performance because image features require dynamic handling. When requested space exceeds the current optimized allocation, latency is incurred while space is expanded.
Encryption	Not applicable.	Optional. Uses 256-bit AES (CRC) cipher or LUKS crypt.
Snapshots	Not applicable.	QCOW2 supports multiple snapshots, each one a read-only record of the image at a point in time. QCOW2 snapshots are stored internally.
Copy-on-write	Not applicable.	QCOW2 file changes are written to a separate overlay file, preserving the original image file, copying the block to be changed to the overlay before modification. Several overlay files can share the same base image. Using multiple files allows running many instances from the same base image.

Your application use case determines the choice of RAW or QCOW2 images. The QCOW2 copy-on-write feature saves significant space when numerous instances are created from the same base image. The advantage increases proportionally with the number of instances deployed per image on each compute host. Although RAW images are preallocated and cannot be shared by multiple instances, CPU processing is significantly reduced due to the lack of additional image handling, and work is not interrupted to add space as when QCOW2 needs to exceed its current allocation. Legacy server configurations and applications that need the greatest performance available are likely to use the RAW format.

IMAGE SIZE CONSIDERATIONS

When you create images in OpenStack, you should set the Minimum Disk (GB) and Minimum RAM (MB) screen fields, or the **`--min-disk`** and **`--min-ram`** command line options with sufficiently sized values. The RAM value should reflect the launched instance's runtime memory requirement. The disk value must be at least the image's virtual size, which was the source disk's original size. When deciding what value to set, keep in mind the difference between an image's logical, physical, and virtual sizes. These values are affected by the image's type and the storage support for sparse files.

A sparse file attempts to use file system space more efficiently when the file is partially empty. File systems that support it will write metadata representing the empty blocks to disk instead of writing empty space in the file. Files can only be sparse when stored in a supporting file system or storage service. Sparse files are supported by `ext2`, `ext3`, `ext4`, XFS, recent NTFS, and APFS file systems, but not `fat32` or HFS+. Ceph RBD also supports sparse file storage.

Commonly, QCOW2 files are referred to as being sparse because they use a similar space-optimization approach. The difference is that a QCOW2 file is optimized internally in its own file structure, as opposed to using the sparse metadata file system or storage service method. Files in QCOW2 format delay storage allocation until needed, and their file-system size represents only non-empty data blocks that are currently allocated. A QCOW2 file's physical and logical sizes are the same, while the virtual size is larger, representing the space needed to recreate the original source file system. The **`ls -s`** command's physical (allocated) size includes file-system pointer blocks not included in the matching `disk size` field of the **`qemu-img`** command.

```
[user@demo ~]$ ls -lsh osp-small.qcow2
1.6G -rw-rw-r--. 1 student student 1.6G Sep 25 17:01 osp-small.qcow2

[user@demo ~]$ qemu-img info osp-small.qcow2
image: osp-small.qcow2
file format: qcow2
virtual size: 10G (10737418240 bytes)
disk size: 1.5G
...output omitted...
```

Conversely, a RAW image is a bit-equivalent copy of the original source file. RAW image file listings display as full size, but are efficiently stored as sparse by the file system or storage back end driver. For a RAW file, the logical size matches the virtual size, but the physical (allocated) size is smaller.

```
[user@demo ~]$ ls -lsh osp-small.raw
1.9G -rw-r--r--. 1 student student 10G Sep 30 09:32 osp-small.raw

[user@demo ~]$ qemu-img info osp-small.raw
image: osp-small.raw
file format: raw
virtual size: 10G (10737418240 bytes)
disk size: 1.8G
...output omitted...
```

Recommended practice is that the **`--min-disk`** and **`--min-ram`** options should not be left blank. Without this guidance, the Compute service will still attempt to launch an instance, even though the flavor may be too small. It will eventually fail when the resulting volume is too small for the image's virtual size. When the options have valid settings, launching an instance with too small a flavor correctly displays an error message without making the attempt.

The image format also affects bandwidth efficiency. Although most storage properly handles sparse files, typical API calls and network file transfers do not. RAW files are processed or transferred at full size, including empty blocks, which is relevant for block (non-shared) live migration and first-time image use, but have the advantage of being exportable to other emulators. The QCOW2 format is preferred for file transfer situations, because QCOW2 files are space-optimized even when using non-sparse file-systems, and take less bandwidth on API calls and network transfers. The QCOW2 format also supports additional **zlib** compression, trading higher CPU utilization for even smaller transfer sizes.

IMAGES IN OPENSTACK SERVICES

The Compute service runs and manages instances within Red Hat OpenStack Platform. The required image format depends on the configured back-end storage system. In the default file-based back end, QCOW2 is preferred because `libvirt` does not support RAW image snapshots. When using Ceph, the image can be converted to RAW format to use the Ceph cluster's snapshot capability.

Run the following command to convert a QCOW2 image file to a RAW image file.

```
[user@demo ~]$ qemu-img convert -f qcow2 -O raw osp-small.qcow2 osp-small.raw
```

When you deploy an instance in a local storage configuration, the image is cached on the compute node as a base image, and then expanded to match the size specified by the flavor. The resized image is used to create the virtual disk for the instance. The default configuration for the compute service converts non-RAW base images to RAW format. This can be disabled if slow I/O or disk space restrictions exist, but CPU use will increase for image handling.

REFERENCES

Further information is available in the documentation for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform

DESCRIBING IMAGE FORMATS

Choose the correct answers to the following questions:

1. What is the correct image format when using Ceph as the back end for the OpenStack Image service?

a. QCOW2
b. VHD
c. VMDK
d. RAW

2. Which four of the following image formats are supported by Red Hat OpenStack Platform? (Choose four.)

a. VMDK
b. FLAR
c. VHD
d. QCOW2
e. RAW

3. Which three features are part of the QCOW2 format? (Choose three.)

a. Encryption
b. DFRWS support
c. Snapshots
d. Multi-bit error correction
e. Copy-on-write

DESCRIBING IMAGE FORMATS

Choose the correct answers to the following questions:

1. What is the correct image format when using Ceph as the back end for the OpenStack Image service?

a. QCOW2
b. VHD
c. VMDK
d. RAW

2. Which four of the following image formats are supported by Red Hat OpenStack Platform? (Choose four.)

a. VMDK
b. FLAR
c. VHD
d. QCOW2
e. RAW

3. Which three features are part of the QCOW2 format? (Choose three.)

a. Encryption
b. DFRWS support
c. Snapshots
d. Multi-bit error correction
e. Copy-on-write

BUILDING CUSTOMIZED IMAGES

OBJECTIVE

After completing this section, students should be able to build an image using diskimage-builder.

BUILDING A CUSTOM IMAGE

The benefits of building custom images include: ensuring that monitoring agents are present, aligning with the organization's security policy, and provisioning a common set of troubleshooting tools. You can use any of three methods to generate a custom image: build an image using **`diskimage-builder`**, customize an existing image using **`guestfish`** or **`virt-customize`**, or use **`cloud-init`** to perform customization after the instance is deployed. Post-deployment customization using a configuration management system such as Red Hat Ansible Automation is beyond the scope of this course. This section discusses **`diskimage-builder`**; the following sections discuss the other two methods.

`diskimage-builder` is a tool for building and customizing cloud images. It can output virtual disk images in a variety of formats, such as QCOW2 and RAW. **`diskimage-builder`** applies *elements* during the build process to customize the image. An element is a code set that runs within a **`chroot`** environment and alters how an image is built. For example, the **`docker`** elements export a tar file from a named container allowing other elements to build on top of it, or the element **`bootloader`**, which installs **`grub2`** on the boot partition of the system.

DISKIMAGE-BUILDER ARCHITECTURE

`diskimage-builder` bind mounts **`/proc`**, **`/sys`**, and **`/dev`** in a **`chroot`** environment. The image-building process produces minimal systems that possess all the required components to fulfill their purpose with OpenStack. Images can be as simple as a file-system image or can be customized to provide whole disk images. Upon completion of the file system tree, a loopback device with file system (or partition table and file system) is built and the file system tree copied into it.

Diskimage-builder Elements

Elements are used to specify what goes into the image and any modifications that are desired. Images are required to use at least one base distribution element, and there are multiple elements for a given distribution. For example, the distribution element could be **`rhel7`**, and then other elements are used to modify the **`rhel7`** base image. Scripts are invoked and applied to the image based on multiple elements.

Diskimage-builder Element Dependencies

Each element can use **`element-deps`** and **`element-provides`** to define or affect dependencies. **`element-deps`** is a plain-text file containing a list of elements that are added to the list of elements built into the image when it is created. **`element-provides`** is a plain-text file that contains a list of elements that are *provided* by this element. These particular elements are *not* included with the elements built into the image at creation time.

The *diskimage-builder* package includes numerous elements:

```
[user@demo ~]$ ls /usr/share/diskimage-builder/elements
apt-conf                        docker                    pip-and-virtualenv
apt-preferences                 dpkg                      pip-cache
```

```
apt-sources                        dracut-network             pkg-map
architecture-emulation-binaries    dracut-ramdisk             posix
baremetal                          dynamic-login              proliant-tools
base                               element-manifest           pypi
bootloader                         enable-serial-console      python-brickclient
cache-url                          epel                       ramdisk
centos                             fedora                     ramdisk-base
centos7                            fedora-minimal             rax-nova-agent
centos-minimal                     gentoo                     redhat-common
cleanup-kernel-initrd              growroot                   rhel
cloud-init                         grub2                      rhel7
cloud-init-datasources             hpdsa                      rhel-common
cloud-init-disable-resizefs        hwburnin                   rpm-distro
cloud-init-nocloud                 hwdiscovery                runtime-ssh-host-keys
debian                             ilo                        select-boot-kernel-
initrd
debian-minimal                     install-bin                selinux-permissive
debian-systemd                     install-static             serial-console
debian-upstart                     install-types              simple-init
debootstrap                        ironic-agent               source-repositories
deploy                             ironic-discoverd-ramdisk   stable-interface-names
deploy-baremetal                   iso                        svc-map
deploy-ironic                      local-config               sysctl
deploy-kexec                       manifests                  uboot
deploy-targetcli                   mellanox                   ubuntu
deploy-tgtadm                      modprobe-blacklist         ubuntu-core
devuser                            no-final-image             ubuntu-minimal
dhcp-all-interfaces                oat-client                 ubuntu-signed
dib-init-system                    openssh-server             vm
dib-python                         opensuse                   yum
dib-run-parts                      opensuse-minimal           yum-minimal
disable-selinux                    package-installs           zypper
dkms                               partitioning-sfdisk        zypper-minimal
```

Each element has scripts that are applied to the images as they are built. The following example shows the scripts for the **base** element.

```
[user@demo ~]$ tree /usr/share/diskimage-builder/elements/base
/usr/share/diskimage-builder/elements/base/
|-- cleanup.d
|   |-- 01-ccache
|   `-- 99-tidy-logs
|-- element-deps
|-- environment.d
|   `-- 10-ccache.bash
|-- extra-data.d
|   `-- 50-store-build-settings
|-- install.d
|   |-- 00-baseline-environment
|   |-- 00-up-to-date
|   |-- 10-cloud-init
|   |-- 50-store-build-settings
|   `-- 80-disable-rfc3041
|-- package-installs.yaml
|-- pkg-map
|-- pre-install.d
```

```
|   `-- 03-baseline-tools
|-- README.rst
`-- root.d
    `-- 01-ccache

    6 directories, 15 files
```

Diskimage-builder Phase Subdirectories

Phase subdirectories should be located under the **`element`** directory; they may or may not exist by default, so create them as required. They contain executable scripts that have a two-digit numerical prefix, and are executed in numerical order. The convention is to store data files in the **`element`** directory, but to only store executable scripts in the phase subdirectory. If a script is not executable it will not run. The phase subdirectories are processed in the order listed in the following table:

Phase Subdirectories

PHASE SUBDIRECTORY	DESCRIPTION
`root.d`	Builds or modifies the initial root file system content. This is where customizations are added, such as building on an existing image. Only one element can use this at a time unless particular care is taken not to overwrite, but instead to adapt the context extracted by other elements. Context extracted by previous elements should be adapted, take care not to overwrite it.
`extra-data.d`	Include extra data from the host environment that hooks may need when building the image. This copies any data such as SSH keys, or HTTP proxy settings, under **`$TMP_HOOKS_PATH`**.
`pre-install.d`	Prior to any customization or package installation, this code runs in a **`chroot`** environment.
`install.d`	In this phase the operating system and packages are installed. This code runs in a **`chroot`** environment.
`post-install.d`	This is the recommended phase to use for performing tasks that must be handled after the operating system and application installation, but before the first boot of the image. For example, running **`systemctl enable`** to enable required services.
`block-device.d`	Customize the block device, for example, to make partitions. Runs before the **`cleanup.d`** phase runs, but after the target tree is fully populated.
`finalize.d`	Runs in a **`chroot`** environment upon completion of the root file-system content being copied to the mounted file system. The root file system is tuned in this phase, so it is important to limit the operations to only those necessary to affect the file system metadata and the image itself. **`post-install.d`** is preferred for most operations.
`cleanup.d`	The root file system content is cleaned of temporary files.

Diskimage-builder Environment Variables

A number of environment variables must be exported, depending upon the required image customization. Typically, at a minimum, the following variables are exported:

Minimal Diskimage-builder Variables

VARIABLE	DESCRIPTION
DIB_LOCAL_IMAGE	The base image to build from.
DIB_YUM_REPO_CONF	The client Yum repository configuration files to be copied into the **chroot** environment during image building.
ELEMENTS_PATH	The path to a working copy of the elements.

IMPORTANT

Yum repository configuration files specified by `DIB_YUM_REPO_CONF` are copied into **/etc/yum.repos.d** during the image build and removed when the build is done. The intention is to provide the specified Yum repository access only during the build and not to leave that Yum repository access in the final image. However, this removal behavior may cause an unintended result; a Yum repository configuration file specified in `DIB_YUM_REPO_CONF` that matches an *already existing* configuration file in the starting base image will result in that configuration file being removed from the final image at the end of the build. Be sure to check for existing repository configuration and exclude it from `DIB_YUM_REPO_CONF` if it should remain in the final built image.

Diskimage-builder Options

This section examines some of the options available in the context of the following example:

```
[user@demo ~]$ disk-image-create vm rhel7 -n \
-p python-django-compressor -a amd64 -o web.img 2>&1 | tee diskimage-build.log
```

The **vm** argument defines the element to use. The **vm** element provides sane defaults for virtual machine disk images. The **rhel7** option specifies that the image will be Red Hat Enterprise Linux 7. The **-n** option skips the default inclusion of the base element, which might be desirable if you prefer not to have *cloud-init* and package updates installed. The **-p** option specifies which packages to install; in this example the *python-django-compressor* package is installed. The **-a** option specifies the architecture of the image. The **-o** option specifies the output image name.

Diskimage-builder Execution

Each element contains a set of scripts to execute. In the following excerpt from the **diskimage-build.log** file, you can see the scripts that were executed as part of the root phase.

```
Target: root.d

Script                                     Seconds
----------------------------------------  ----------

01-ccache                                     0.017
10-rhel7-cloud-image                         93.202
50-yum-cache                                  0.045
```

```
90-base-dib-run-parts                          0.037
```

The run time for each script is shown on the right. Scripts that reside in the **`extra-data.d`** phase subdirectory were then executed:

```
Target: extra-data.d

Script                                      Seconds
---------------------------------------  ----------

01-inject-ramdisk-build-files                  0.031
10-create-pkg-map-dir                          0.114
20-manifest-dir                                0.021
50-add-targetcli-module                        0.038
50-store-build-settings                        0.006
75-inject-element-manifest                     0.040
98-source-repositories                         0.041
99-enable-install-types                        0.023
99-squash-package-install                      0.221
99-yum-repo-conf                               0.039
```

From these examples, you can confirm the order that the phases were executed in and the order of script execution in each phase.

REFERENCES

Diskimage-builder Documentation

https://docs.openstack.org/diskimage-builder/latest/

BUILDING AN IMAGE

In this exercise you will build and customize a disk image using **diskimage-builder**.

OUTCOMES

You should be able to:

- Build and customize an image using **diskimage-builder**.
- Upload the image into the OpenStack image service.
- Launch an instance using the customized image.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab customization-img-building setup`** command. This ensures that the required packages are installed on `workstation`, and provisions the environment with a public network, a private network, a key pair, and security rules to access the instance.

```
[student@workstation ~]$ lab customization-img-building setup
```

1. On `workstation`, retrieve the `http://materials.example.com/osp-small.qcow2` disk image and save it under **`/home/student/`**.

```
[student@workstation ~]$ wget http://materials.example.com/osp-small.qcow2
```

2. Create a copy of the **`/usr/share/diskimage-builder/elements`** directory to work with under **`/home/student/`**.

```
[student@workstation ~]$ cp -a /usr/share/diskimage-builder/elements \
/home/student/
```

3. Create a **`post-install.d`** directory under the working copy of the **`rhel7`** element.

```
[student@workstation ~]$ mkdir -p /home/student/elements/rhel7/post-install.d
```

4. Add two scripts under the **`rhel7`** element **`post-install.d`** directory to enable the named service, and add a **`forwarders`** option to **`/etc/named.conf`**.

```
[student@workstation ~]$ cd /home/student/elements/rhel7/post-install.d/
[student@workstation post-install.d]$ cat <<EOF > 01-enable-services
#!/bin/bash
systemctl enable named
```

```
EOF
[student@workstation post-install.d]$ cat <<EOF > 02-configure-named
#!/bin/bash

# add forwarders
sed -i 's|\(.*\)\(recursion yes;\)|\1\2\n\1forwarders {172.25.250.254;};|' \
/etc/named.conf

# allow queries from the local subnet
sed -i 's|\(.*allow-query\).*|\1     { localhost; 192.168.1.0/24; };|' \
/etc/named.conf

# disable dnssec validation
sed -i 's|\(.*dnssec-validation\).*|\1 no;|' /etc/named.conf
EOF
```

5. Change to the **/home/student** directory and set the executable permission on the scripts.

```
[student@workstation post-install.d]$ cd
[student@workstation ~]$ chmod +x /home/student/elements/rhel7/post-install.d/*
```

6. Export the following environment variables, which diskimage-builder requires. The DIB_NO_TMPFS variable is only required because workstation has a small hard disk, it is not required in most cases.

Environment Variables

VARIABLE	CONTENT
DIB_LOCAL_IMAGE	**/home/student/osp-small.qcow2**
DIB_YUM_REPO_CONF	**/etc/yum.repos.d/openstack.repo**
ELEMENTS_PATH	**/home/student/elements**
DIB_NO_TMPFS	**1**

```
[student@workstation ~]$ export DIB_LOCAL_IMAGE=/home/student/osp-small.qcow2
[student@workstation ~]$ export DIB_YUM_REPO_CONF=/etc/yum.repos.d/openstack.repo
[student@workstation ~]$ export ELEMENTS_PATH=/home/student/elements
[student@workstation ~]$ export DIB_NO_TMPFS=1
```

7. Build the **finance-rhel-dns.qcow2** disk image and include the *bind* and *bind-utils* packages. The scripts created earlier are automatically integrated. Enter the password for student when prompted.

```
[student@workstation ~]$ disk-image-create vm rhel7 \
-t qcow2 \
-p bind,bind-utils \
-o finance-rhel-dns.qcow2
...output omitted...
```

8. As the `developer1` OpenStack user, upload the **`finance-rhel-dns.qcow2`** disk image to the OpenStack Image service as `finance-rhel-dns`, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.

8.1. Source the **`developer1-finance-rc`** credentials file.

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

8.2. Using the **`openstack`** command, upload the disk image **`finance-rhel-dns.qcow2`** into the OpenStack Image service.

```
[student@workstation ~(developer1-finance)]$ openstack image create \
--disk-format qcow2 \
--min-disk 10 \
--min-ram 2048 \
--file ~/finance-rhel-dns.qcow2 \
finance-rhel-dns
...output omitted...
```

9. Launch an instance in the environment using the following attributes.

Instance Attributes

ATTRIBUTE	VALUE
flavor	default
key pair	example-keypair
network	finance-network1
image	finance-rhel-dns
security group	finance-dns
name	finance-dns1

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=finance-network1 \
--image finance-rhel-dns \
--security-group finance-dns \
--wait finance-dns1
...output omitted...
```

10. List the available floating IP addresses, and then allocate one to `finance-dns1`.

10.1. List the available floating IPs. The floating IPs available on your system may differ from those shown below.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip list \
-c "Floating IP Address" -c Port
```

```
+---------------------+------+
| Floating IP Address | Port |
+---------------------+------+
| 172.25.250.110      | None |
| 172.25.250.107      | None |
+---------------------+------+
```

10.2. Attach an available floating IP to the `finance-dns1` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server add floating \
ip finance-dns1 172.25.250.110
```

11. If the image build was successful, the resulting DNS server should accept queries and forward them to the Domain Name Service running on `workstation`.

11.1. Use the **ssh** command to log in to `finance-dns1` as `cloud-user`.

```
[student@workstation ~(developer1-finance)]$ ssh cloud-user@172.25.250.110
Warning: Permanently added '172.25.250.110' (ECDSA) to the list of known hosts.
[cloud-user@finance-dns1 ~]$
```

11.2. Ensure that `named` is listening on port 53.

```
[cloud-user@finance-dns1 ~]$ sudo ss -4nlp | grep named
udp    UNCONN  0  0    127.0.0.1:53    *:*   users:(("named",pid=860,fd=513),...
tcp    LISTEN  0  10   127.0.0.1:53    *:*   users:(("named",pid=860,fd=21))
tcp    LISTEN  0  128  127.0.0.1:953   *:*   users:(("named",pid=860,fd=23))
```

11.3. Use the **dig** command to perform a query against the local `named` service. Query the NS records for the `example.com` domain.

```
[cloud-user@finance-dns1 ~]$ dig @localhost example.com NS

; <<>> DiG 9.9.4-RedHat-9.9.4-61.el7 <<>> @localhost example.com NS
; (2 servers found)
;; global options: +cmd
;; Got answer:
;; ->>HEADER<<- opcode: QUERY, status: NOERROR, id: 43472
;; flags: qr rd ra; QUERY: 1, ANSWER: 1, AUTHORITY: 0, ADDITIONAL: 2

;; OPT PSEUDOSECTION:
; EDNS: version: 0, flags:; udp: 4096
;; QUESTION SECTION:
;example.com.   IN NS

;; ANSWER SECTION:
example.com.  85612 IN NS classroom.example.com.

;; ADDITIONAL SECTION:
classroom.example.com. 85612 IN A 172.25.254.254

;; Query time: 1 msec
;; SERVER: ::1#53(::1)
;; WHEN: Thu Sep 20 04:41:28 EDT 2018
;; MSG SIZE  rcvd: 80
```

```
[cloud-user@finance-dns1 ~]$ logout
[student@workstation ~(developer1-finance)]$
```

Note that there is no **aa** flag in the flags section of the output, this means the server that we queried was not authoritative for the `example.com` domain. Therefore the query was answered by an upstream name server and `finance-dns1` must be working as designed.

Cleanup

On `workstation`, run the **`lab customization-img-building cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab customization-img-building cleanup
```

MODIFYING GUEST AND DISK IMAGES

OBJECTIVES

After completing this section, students should be able to customize an image using **guestfish** and **virt-customize**.

MAKING MINOR IMAGE CUSTOMIZATIONS

Building an image using **diskimage-builder** can take several minutes, and may require a copy of the **elements** directory to be used for each image. If you only require a small number of customizations, you could save time by using the **guestfish** or **virt-customize** commands to modify a base image, such as the one provided by Red Hat in the *rhel-guest-image-7* package. This image has a minimal set of packages installed, and **Cloud-init** installed and enabled. You can download the *rhel-guest-image-7* package from `https://access.redhat.com/downloads`.

GUESTFISH AND VIRT-CUSTOMIZE INTERNALS

guestfish and **virt-customize** both use the *libguestfs* API to perform their functions. libguestfs needs a back end that can work with the various image formats, and by default it uses **libvirt**. The process for editing an image with a **libvirt** back end includes creating an overlay file for the image, creating an appliance, booting the appliance with or without network support, and mounting the partitions. You can investigate the process in more detail by exporting two environment variables, `LIBGUESTFS_DEBUG=1` and `LIBGUESTFS_TRACE=1`, and then executing **guestfish** or **virt-customize** with the **-a** option to add a disk.

USING GUESTFISH TO CUSTOMIZE IMAGES

guestfish is a low-level tool that exposes the libguestfs API directly, which means that you can manipulate images in a very granular fashion. The Guestfish shell is not a Bash shell, however it does offer some similar commands such as **cat**, **mount**, and **mkfs**. The following example uses the **-i** option to mount partitions automatically, the **-a** option to add the disk image, and the **--network** option to enable network access. It then installs the *aide* package, sets the password for the `root` user, and restores SELinux file contexts.

```
[user@demo ~]$ guestfish -i --network -a ~/demo-rhel-base.qcow2

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

Operating system: Red Hat Enterprise Linux Server 7.3 (Maipo)
/dev/sda1 mounted on /

><fs> command "yum -y install aide"
...output omitted...
><fs> command "echo letmein | passwd --stdin root"
><fs> selinux-relabel /etc/selinux/targeted/contexts/files/file_contexts /
```

USING VIRT-CUSTOMIZE TO CUSTOMIZE IMAGES

`virt-customize` is a high-level tool that also uses the libguestfs API, but eases image building by performing tasks using simple options that may have required multiple API calls to achieve using **`guestfish`** or the **`libguestfs`** API directly. The following example shows **`virt-customize`** using the **`-a`** option to add the disk, installing a package, setting the `root` password, and resetting SELinux contexts.

```
[user@demo ~]$ virt-customize -a ~/demo-rhel-base.qcow2 \
--install aide \
--root-password password:letmein \
--selinux-relabel
...output omitted...
```

The following table compares the two tools.

Comparison of guestfish and virt-customize Commands

FEATURE	GUESTFISH	VIRT-CUSTOMIZE
Complexity	A low-level tool that exposes the guestfs API directly.	A high-level tool that is easier to use and simplifies common tasks.
SELinux support	Use the **`selinux-relabel /etc/selinux/targeted/contexts/files/file_contexts /`** command to restore SELinux file contexts.	Use the **`--selinux-relabel`** option to restore file contexts. This option uses the **`touch /.autorelabel`** command if relabeling is unsuccessful.
Options	For low-level tasks such as manipulating partitions, scripting, and remote access.	For common tasks such as installing packages, changing passwords, setting the host name and time zone, and registering with Subscription Manager.

The **`--selinux-relabel`** customization option relabels files in the guest so that they have the correct SELinux label. This option tries to relabel files immediately. If unsuccessful, **`/.autorelabel`** is created on the image. This schedules the relabel operation for the next time the image boots.

IMPORTANT

Files modified from inside the guestfish tool are written without a valid SELinux context. Failure to relabel critical modified files, such as **`/etc/passwd`**, will result in an unusable image, because SELinux properly denies access to files with improper context during the boot process.

Although a relabel can be configured using **`touch /.autorelabel`** from within **`guestfish`**, this would be persistent in the image, resulting in a relabel being performed on first boot for every instance deployed using this image.

USE CASES

For most common image customization tasks, **`virt-customize`** is the best choice. However, as listed in the table above, the less frequent low-level tasks should be performed with the **`guestfish`** command.

ADDITIONAL TOOLS

`virt-builder` is another tool in the libguestfs suite that includes the ability to download, build, and customize images. **`virt-customize`** was created by taking the customization parts of **`virt-builder`** and making a standalone tool. **`virt-sysprep`** is a tool that was designed to deconfigure a virtual machine by removing host-specific information such as SSH host keys, log files, and so on. It only operates on offline disk images, whereas the other tools boot an instance, customize it, and then save it back to disk.

REFERENCES

guestfish - the guest file system shell
http://libguestfs.org/guestfish.1.html

virt-customize - Customize a virtual machine
http://libguestfs.org/virt-customize.1.html

virt-builder - Build virtual machine images quickly
http://libguestfs.org/virt-builder.1.html

virt-sysprep - Reset, unconfigure or customize a virtual machine so clones can be made
http://libguestfs.org/virt-sysprep.1.html

CUSTOMIZING AN IMAGE

In this exercise you will customize disk images using **guestfish** and **virt-customize**.

OUTCOMES

You should be able to:

- Customize an image using **guestfish**.
- Customize an image using **virt-customize**.
- Upload an image into Glance.
- Spawn an instance using a customized image.

BEFORE YOU BEGIN

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

On workstation, run the **lab customization-img-customizing setup** command. This ensures that the required packages are installed on workstation, and provisions the environment with a public network, a private network, a private key, and security rules to access the instance.

```
[student@workstation ~]$ lab customization-img-customizing setup
```

1. On workstation, retrieve the **osp-small.qcow2** disk image from http://materials.example.com/osp-small.qcow2 and save it as **/home/student/finance-rhel-db.qcow2**.

```
[student@workstation ~]$ wget http://materials.example.com/osp-small.qcow2 \
-O ~/finance-rhel-db.qcow2
```

2. Use the **guestfish** command to update the image to include network access.

```
[student@workstation ~]$ guestfish -i --network -a ~/finance-rhel-db.qcow2

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
      'man' to read the manual
      'quit' to quit the shell

Operating system: Red Hat Enterprise Linux Server 7.5 (Maipo)
/dev/sda1 mounted on /

><fs>
```

3. Install the *mariadb* and *mariadb-server* packages.

```
><fs> command "yum -y install mariadb mariadb-server"
...output omitted...
Installed:
  mariadb.x86_64 3:10.1.20-2.el7ost   mariadb-server.x86_64 3:10.1.20-2.el7ost

Dependency Installed:
  libaio.x86_64 0:0.3.109-13.el7
  lsof.x86_64 0:4.87-5.el7
  mariadb-errmsg.x86_64 3:10.1.20-2.el7ost
  perl-Compress-Raw-Bzip2.x86_64 0:2.061-3.el7
  perl-Compress-Raw-Zlib.x86_64 1:2.061-4.el7
  perl-DBD-MySQL.x86_64 0:4.023-6.el7
  perl-DBI.x86_64 0:1.627-4.el7
  perl-Data-Dumper.x86_64 0:2.145-3.el7
  perl-IO-Compress.noarch 0:2.061-2.el7
  perl-Net-Daemon.noarch 0:0.48-5.el7
  perl-PlRPC.noarch 0:0.2020-14.el7
  socat.x86_64 0:1.7.3.2-2.el7

Complete!
```

4. Enable the `mariadb` service.

```
><fs> command "systemctl enable mariadb"
```

5. Because there was no output, ensure the `mariadb` service was enabled.

```
><fs> command "systemctl is-enabled mariadb"
enabled
```

6. Ensure the SELinux contexts for all affected files are correct.

IMPORTANT
Do not forget to fix SELinux file contexts at this time.

```
><fs> selinux-relabel /etc/selinux/targeted/contexts/files/file_contexts /
```

7. Log out from the **`guestfish`** shell.

```
><fs> exit

[student@workstation ~]$
```

8. As the `developer1` OpenStack user, upload the **`finance-rhel-db.qcow2`** image to the image service as **`finance-rhel-db`**, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.

8.1. Source the **`/home/student/developer1-finance-rc`** credential file.

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

8.2. As the `developer1` OpenStack user, upload the **`finance-rhel-db.qcow2`** image to the image service as `finance-rhel-db`.

```
[student@workstation ~(developer1-finance)]$ openstack image create \
--disk-format qcow2 \
--min-disk 10 \
--min-ram 2048 \
--file ~/finance-rhel-db.qcow2 \
finance-rhel-db
...output omitted...
```

9. Launch an instance in the environment using the following attributes:

Instance Attributes

ATTRIBUTE	VALUE
flavor	default
key pair	example-keypair
network	finance-network1
image	finance-rhel-db
security group	finance-db
name	finance-db1

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=finance-network1 \
--security-group finance-db \
--image finance-rhel-db \
--wait finance-db1
...output omitted...
```

10. List the available floating IP addresses, and then allocate one to `finance-db1`.

10.1. List the floating IPs; unallocated IPs have **None** listed as their **Port** value.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip list \
-c "Floating IP Address" -c Port
+---------------------+------+
```

```
| Floating IP Address | Port |
+---------------------+------+
| 172.25.250.110      | None |
| 172.25.250.107      | None |
+---------------------+------+
```

10.2. Attach an unallocated floating IP to the `finance-db1` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server add floating \
ip finance-db1 172.25.250.110
```

11. Use **ssh** to connect to the `finance-db1` instance. Ensure the *mariadb-server* package is installed, and that the `mariadb` service is enabled and running.

11.1. Use the **ssh** command to log in to `finance-db1` as the `cloud-user` user.

```
[student@workstation ~(developer1-finance)]$ ssh cloud-user@172.25.250.110
Warning: Permanently added '172.25.250.110' (ECDSA) to the list of known hosts.
[cloud-user@finance-db1 ~]$
```

11.2. Confirm that the *mariadb-server* package is installed.

```
[cloud-user@finance-db1 ~]$ rpm -q mariadb-server
mariadb-server-10.1.20-2.el7ost.x86_64
```

11.3. Confirm that the `mariadb` service is enabled and running, and then log off.

```
[cloud-user@finance-db1 ~]$ systemctl status mariadb
...output omitted...
   Loaded: loaded (/usr/lib/systemd/system/mariadb.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Wed 2018-09-19 04:21:31 EDT; 1h 38min ago
  Process: 1033 ExecStartPost=/usr/libexec/mariadb-wait-ready $MAINPID
 (code=exited, status=0/SUCCESS)
  Process: 815 ExecStartPre=/usr/libexec/mariadb-prepare-db-dir %n (code=exited,
 status=0/SUCCESS)
 Main PID: 1031 (mysqld_safe)
...output omitted...
[cloud-user@finance-db1 ~]$ logout
Connection to 172.25.250.110 closed.
[student@workstation ~(developer1-finance)]$
```

12. On `workstation`, retrieve the **`osp-small.qcow2`** image from `http://materials.example.com/osp-small.qcow2` and save it as **`/home/student/finance-rhel-mail.qcow2`**.

```
[student@workstation ~(developer1-finance)]$ wget \
http://materials.example.com/osp-small.qcow2 -O ~/finance-rhel-mail.qcow2
```

13. Use the **`virt-customize`** command to customize the **`~/finance-rhel-mail.qcow2`** image. Enable the `postfix` service, configure `postfix` to listen on all interfaces, and relay

all mail to workstation.lab.example.com. Install the *mailx* package to enable sending a test email. Ensure the SELinux contexts are restored.

```
[student@workstation ~(developer1-finance)]$ virt-customize \
-a ~/finance-rhel-mail.qcow2 \
--run-command 'systemctl enable postfix' \
--run-command 'postconf -e "relayhost = [workstation.lab.example.com]"' \
--run-command 'postconf -e "inet_interfaces = all"' \
--run-command 'yum -y install mailx' \
--selinux-relabel
[   0.0] Examining the guest ...
[  26.8] Setting a random seed
[  26.9] Running: systemctl enable postfix
[  27.9] Running: postconf -e "relayhost = [workstation.lab.example.com]"
[  28.8] Running: postconf -e "inet_interfaces = all"
[  29.7] Running: yum -y install mailx
[  54.5] SELinux relabelling
[ 187.8] Finishing off
```

14. As the `developer1` OpenStack user, upload the **`finance-rhel-mail.qcow2`** image to the image service as `finance-rhel-mail`, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.

14.1. Use the **`openstack`** command to upload the **`finance-rhel-mail.qcow2`** image to the image service.

```
[student@workstation ~(developer1-finance)]$ openstack image create \
--disk-format qcow2 \
--min-disk 10 \
--min-ram 2048 \
--file ~/finance-rhel-mail.qcow2 \
finance-rhel-mail
...output omitted...
```

15. Launch an instance in the environment using the following attributes:

Instance Attributes

ATTRIBUTE	VALUE
flavor	default
key pair	example-keypair
network	finance-network1
image	finance-rhel-mail
security group	finance-mail
name	finance-mail1

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
```

```
--key-name example-keypair \
--nic net-id=finance-network1 \
--security-group finance-mail \
--image finance-rhel-mail \
--wait finance-mail1
...output omitted...
```

16. List the available floating IP addresses, and allocate one to `finance-mail1`.

16.1. List the available floating IP addresses.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip list \
-c "Floating IP Address" -c Port
+---------------------+--------------------------------------+
| Floating IP Address | Port                                 |
+---------------------+--------------------------------------+
| 172.25.250.110      | 1ce9ffa5-b52b-4581-a696-52f464912500 |
| 172.25.250.107      | None                                 |
+---------------------+--------------------------------------+
```

16.2. Attach an available floating IP to the `finance-mail1` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server add floating \
ip finance-mail1 172.25.250.107
```

17. Use **ssh** to connect to the `finance-mail1` instance. Ensure the `postfix` service is running, that `postfix` is listening on all interfaces, and that the **`relay_host`** option is correct.

17.1. Use the **ssh** command to log in to the `finance-mail1` instance as the `cloud-user` user.

```
[student@workstation ~(developer1-finance)]$ ssh cloud-user@172.25.250.107
Warning: Permanently added '172.25.250.107' (ECDSA) to the list of known hosts.
[cloud-user@finance-mail1 ~]$
```

17.2. Ensure the `postfix` service is running.

```
[cloud-user@finance-mail1 ~]$ systemctl status postfix
...output omitted...
   Loaded: loaded (/usr/lib/systemd/system/postfix.service; enabled; vendor
 preset: disabled)
   Active: active (running) since Wed 2018-09-19 06:12:00 EDT; 8min ago
  Process: 1064 ExecStart=/usr/sbin/postfix start (code=exited, status=0/SUCCESS)
  Process: 1061 ExecStartPre=/usr/libexec/postfix/chroot-update (code=exited,
 status=0/SUCCESS)
  Process: 1058 ExecStartPre=/usr/libexec/postfix/aliasesdb (code=exited,
 status=0/SUCCESS)
 Main PID: 1136 (master)
...output omitted...
```

17.3. Ensure `postfix` is listening on all interfaces.

```
[cloud-user@finance-mail1 ~]$ sudo ss -tnlp | grep master
LISTEN     0    100       *:25       *:*      users:(("master",pid=1136,fd=13))
```

```
LISTEN     0    100      :::25      :::*      users:(("master",pid=1136,fd=14))
```

17.4. Ensure the **relayhost** option is correctly configured.

```
[cloud-user@finance-mail1 ~]$ postconf relayhost
relayhost = [workstation.lab.example.com]
```

17.5. Send a test email to `<student@workstation.lab.example.com>`.

```
[cloud-user@finance-mail1 ~]$ mail -s "Test" student@workstation.lab.example.com
Hello World!
.
EOT
```

17.6. Return to `workstation`. Use the **mail** command to confirm that the test email arrived.

```
[cloud-user@finance-mail1 ~]$ logout
Connection to 172.25.250.107 closed.
You have new mail in /var/spool/mail/student
[student@workstation ~]$ mail
Heirloom Mail version 12.5 7/5/10.  Type ? for help.
"/var/spool/mail/student": 1 message 1 new
>N  1 Cloud User            Wed Sep 19 10:21  21/938   "Test"
& q
Held 1 message in /var/spool/mail/student
You have mail in /var/spool/mail/student
[student@workstation ~]$
```

Cleanup

On `workstation`, run the **lab customization-img-customizing cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab customization-img-customizing cleanup
```

INITIALIZING AN INSTANCE DURING DEPLOYMENT

OBJECTIVES

After completing this section, students should be able to customize launched instances during deployment using **`cloud-init`**.

REVIEWING CLOUD-INIT BASICS

To avoid a proliferation of images, you can maintain just a few generic base images, and then perform instance-specific customization during deployment. **`cloud-init`** is a set of Python scripts and utilities that perform early initialization of instances. Its tasks may include injecting SSH keys, configuring network settings, and setting the host name. The information required for **`cloud-init`** to configure an instance is known as user-data. There are several formats of user-data supported, including shell script, **`cloud config`** format, upstart job, MIME multi-part format, and **`include`** file. These can all be compressed using **`gzip`**.

Popular Cloud-Init User-data Formats

shell script
: Cloud-init can use shell scripts directly without modification as user-data. This provides an easy migration path if your existing hardening or customization is performed using shell scripts.

cloud config
: The **`cloud config`** format allows administrators to perform common tasks using simple instructions. A **`cloud config`** format file must be valid YAML, and have a **`#cloud-config`** header or **`Content-Type: text/cloud-config`** header for MIME archives. It supports creating users, groups, files, and Yum repositories, installing packages, configuring SSH keys, and other tasks.

include
: An include file contains a list of URLs, each one pointing to user data in any supported format. It must have an **`#include`** header or a **`Content-Type: text/x-include-url`** header for MIME archives.

DESCRIBING CONFIG DRIVE

In Red Hat OpenStack Platform, **`cloud-init`** runs during instance boot and receives the user-data information from the Compute Metadata API. The OpenStack Compute service provides the **`--config-drive`** Boolean option to automatically generate a small ISO 9660- or VFAT-formatted image containing the user-data specific to the instance. This option can provide the instance with static network configuration information if the attached project network does not have DHCP enabled. The generated image has a volume label of **`config-2`** and is attached to the instance when it boots.

The Bare Metal service also supports a **`--config-drive`** option, however this needs a path to a directory containing the files you want to include. The Bare Metal service creates a gzip compressed, Base64-encoded ISO 9660 image with the **`config-2`** label, containing the files in the directory.

DESCRIBING THE METADATA SERVICE ARCHITECTURE

With the introduction of OVN in Red Hat OpenStack Platform 13, the Metadata architecture has changed. The Metadata architecture is now composed of the Metadata Proxy, the Neutron OVN Metadata Agent, and the Nova Metadata API.

DESCRIBING CLOUD-INIT COMMUNICATION WITH THE COMPUTE METADATA API

`cloud-init` running on an instance is directed to the Metadata service by either of two methods. On project networks with DHCP enabled, DHCP responses include a static route for 169.254.169.254. Project networks without DHCP include a static route within the OVN router. The following list describes the communication process.

1. During instance deployment, a compute node is chosen as a deployment target. The `neutron-ovn-metadata-agent` running on the target compute node is responsible for managing network namespaces (ovnmeta-xxx), OVS interfaces, and launching HAProxy processes.

2. If a network namespace (ovnmeta-xxx) for the instance network does not exist, the `neutron-ovn-metadata-agent` creates one.

3. The namespace is then linked to OVS using a veth pair, configured with IP addresses, and brought up.

4. The `neutron-ovn-metadata-agent` then launches an HAProxy process known as a metadata proxy within that namespace. The output below shows the namespace, the interface linking the namespace to OVN, and the HAProxy process all using the same identifier.

```
[root@compute ~]# ip netns
ovnmeta-99c1f7fc-3f92-407f-af59-313b284a43d6 (id: 0)
[root@compute ~]# ovs-vsctl show
...output omitted...
    Bridge br-int
        fail_mode: secure
        Port br-int
            Interface br-int
                type: internal
...output omitted...
        Port "tap99c1f7fc-30"
            Interface "tap99c1f7fc-30"
...output omitted...
[root@compute ~]# pgrep -a haproxy
11468 haproxy -f /var/lib/neutron/ovn-metadata-proxy/99c1f7fc-3f92-407f-
af59-313b284a43d6.conf
```

5. Requests from **`cloud-init`** are sent directly to the metadata proxy if the instance has a host route provided by DHCP, or forwarded to the metadata proxy by the OVN router if DHCP is not enabled on the network.

6. The metadata proxy process listens for HTTP traffic, then forwards any requests to `neutron-ovn-metadata-agent` agent through the **`/var/lib/neutron/metadata_proxy`** socket. As the Compute Metadata API cannot differentiate instances by IP address only, the metadata proxy injects an X-OVN-Network-ID header containing the ID of the project network.

```
listen listener
```

```
    bind 0.0.0.0:80
    server metadata /var/lib/neutron/metadata_proxy
    http-request add-header X-OVN-Network-ID 99c1f7fc-3f92-407f-af59-313b284a43d6
```

7. `neutron-ovn-metadata-agent` then sends the request to the Compute Metadata API, and forwards any responses back over the same path to the instance.

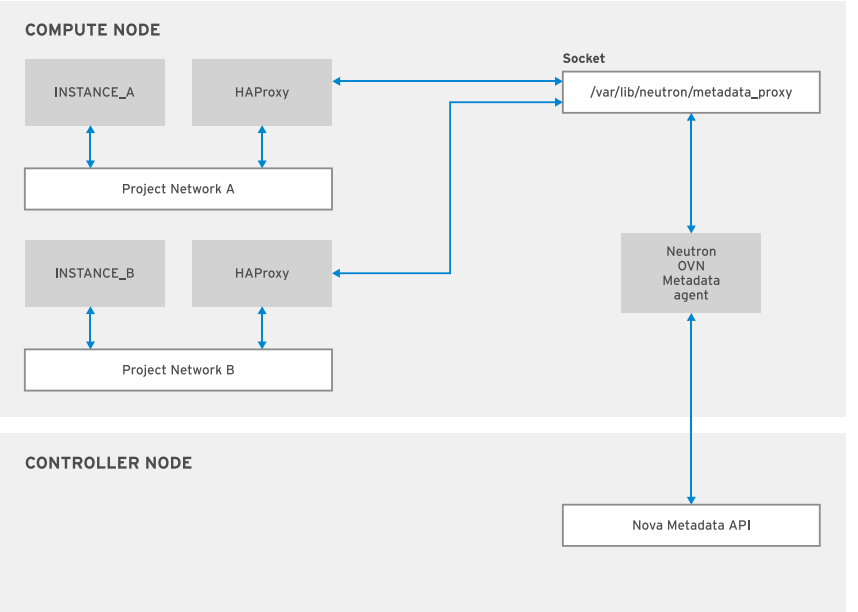


Figure 4.1: Cloud-Init communication to the Compute Metadata API

DESCRIBE CLOUD-INIT STRUCTURE

The **`cloud-init`** application is composed of modules written in Python. It also includes four Systemd services, configuration files, and some templates for distribution-specific files. Data for **`cloud-init`** is stored under **`/var/lib/cloud`**. These files are all present on an instance with the *cloud-init* package installed. The functions of the four services are described below.

Cloud-Init Services

cloud-init-local.service
: This service runs first before the network is available, but when the root partition is mounted read/write. This service is ideally placed to provide network configuration information to the instance. The command executed is **`cloud-init init --local`**.

cloud-init.service
: This service is second in the boot order, and expects the network to be configured. Because the network is available, this stage processes all user-data. All modules listed under **`cloud_init_modules`** in **`/etc/cloud/cloud.conf`** are executed. The command executed is **`cloud-init init`**.

cloud-config.service
: This service runs after `cloud-init.service`. All modules listed under **`cloud_config_modules`** in **`/etc/cloud/cloud.conf`** are executed. The command executed is **`cloud-init modules --mode=config`**.

cloud-final.service
: This service runs as late as possible by depending on the `cloud-config.service` and `rc-local.service` services. All modules listed under **`cloud_final_modules`** in **`/etc/`**

`cloud/cloud.conf` are executed. The command executed is **`cloud-init modules --mode=final`**.

You can troubleshoot **`cloud-init`** by inspecting the log files under **`/var/log/`**. The **`cloud-init.log`** file contains information produced by the four services. The **`cloud-init-output.log`** file may exist, and contains the output produced by each module. Any of the commands listed in the services above can be run manually to troubleshoot issues.

The **`/etc/cloud/cloud.cfg`** file is the primary configuration file for **`cloud-init`**, however additional configuration can be included by placing files under **`/etc/cloud/cloud.cfg.d/`**.

REFERENCES

Cloud-Init - Documentation

https://cloudinit.readthedocs.io/en/latest/index.html

INITIALIZING AN INSTANCE DURING DEPLOYMENT

In this exercise you will customize a launched instance during deployment using **`cloud-init`**.

OUTCOMES

You should be able to:

- Create a user-data file in cloud config format.
- Launch an instance and verify that the user-data has been applied.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab customization-img-cloudinit setup`** command. This ensures that the required packages are installed on `workstation`, and provisions the environment with resources required to launch an instance.

```
[student@workstation ~]$ lab customization-img-cloudinit setup
```

1. Create the **`/home/student/user-data`** file in cloud config format.
   1.1. Edit the **`/home/student/user-data`** file. Add directives to create the `cloud-admin` user and populate the **`/home/cloud-admin/.ssh/authorized_keys`** file with the content of **`/home/student/.ssh/example-keypair.pub`**. Give `cloud-admin` passwordless sudo access. Ensure you include the **`#cloud-config`** header.

NOTE

The output below has been formatted for readability; the SSH public key data should not contain new-line characters.

```
#cloud-config

users:
  - name: cloud-admin
    sudo: ALL=(ALL) NOPASSWD:ALL
    ssh_authorized_keys:
      - ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAABAQDHufCmzG95rJkXHZOe+7rHMaf3me7geA
                fdYAc2fuoKjSVoBni4aF4MYkSmb1UYyXjtQ++x2+i13+Osn9FJZmvda0maqT6D
                uIcpiAxldVrnNJFv5L1VFGiFDbUglThPc25Ytn7bWqg02pFJz4Nc9vN+PzVETe
                vL8b0tMvWPQ44MRuXCCM+UaHO1mBD2pcEnpQ1R/MzYxTzzdvjP5iBn4GAp7KjU
                w/+FvBhlNiKsXJjQGl6MHbCZhtgsntJzl7tKGY4SgJXZaUD0TnpPeCBlEmWNUz
```

```
                4hAoVMfiiZA4fLAmJ7yZqwRcr4EVnqbmZC6CfIhVxb1J69UHhK62KS6513MP3v
                Generated-by-Nova
```

1.2. Ensure the *aide*, *nmap*, *socat*, and *wireshark* packages will be installed.

```
packages:
  - aide
  - nmap
  - socat
  - wireshark
```

1.3. Source **/home/student/.bashrc** to enable the yv alias. Validate that the **/home/student/user-data** file is in YAML format, and correct any reported errors.

```
[student@workstation ~]$ source ~/.bashrc
[student@workstation ~]$ yv ~/user-data
```

2. Source the **/home/student/developer1-finance-rc** environment file. Launch an instance using the following attributes:

Instance Attributes

ATTRIBUTE	VALUE
flavor	**default**
key pair	**example-keypair**
network	**finance-network1**
image	**rhel7**
security group	**default**
user-data	**/home/student/user-data**
name	**finance-admin1**

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=finance-network1 \
--security-group default \
--image rhel7 \
--user-data ~/user-data \
--wait finance-admin1
...output omitted...
```

3. List the available floating IP addresses, and then allocate one to `finance-admin1`.

   3.1. List the floating IPs; unallocated IPs have **None** listed as their **Port** value. Note that your list of floating IPs may vary from the output below in address and quantity.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip list \
-c "Floating IP Address" -c Port
+---------------------+------+
| Floating IP Address | Port |
+---------------------+------+
| 172.25.250.110      | None |
| 172.25.250.107      | None |
+---------------------+------+
```

   3.2. Attach an unallocated floating IP to the `finance-admin1` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server add floating \
ip finance-admin1 172.25.250.110
```

4. Verify that the **`cloud-init`** customization has worked. Debug any failures by checking the **`/var/log/cloud-init.log`** file.

   4.1. Use the **ssh** command to attempt to log in to the `finance-admin1` instance as the `cloud-admin` user. The **`/home/student/.ssh/example-keypair`** private key is used automatically in the classroom.

```
[student@workstation ~(developer1-finance)]$ ssh cloud-admin@172.25.250.110
Warning: Permanently added '172.25.250.110' (ECDSA) to the list of known hosts.
[cloud-admin@finance-admin1 ~]$
```

   4.2. Verify that the *aide*, *nmap*, *socat*, and *wireshark* packages are installed.

```
[cloud-admin@finance-admin1 ~]$ rpm -qa | egrep 'aide|nmap|socat|wireshark'
socat-1.7.3.2-2.el7.x86_64
wireshark-1.10.14-14.el7.x86_64
aide-0.15.1-13.el7.x86_64
nmap-ncat-6.40-13.el7.x86_64
nmap-6.40-13.el7.x86_64
[cloud-admin@finance-admin1 ~]$ logout
Connection to 172.25.250.110 closed.
[student@workstation ~(developer1-finance)]$
```

Cleanup

On `workstation`, run the **`lab customization-img-cloudinit cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab customization-img-cloudinit cleanup
```

PERFORMING IMAGE OPERATIONS

PERFORMANCE CHECKLIST

In this lab, you will build a disk image using diskimage-builder, modify it using **guestfish**, upload it to the Image service, then launch an instance using the new image.

OUTCOMES

You should be able to:

- Build an image using diskimage-builder.
- Customize the image using a post-install script.
- Customize the image using the **guestfish** command.
- Upload the image to the OpenStack Image service.
- Launch an instance using the customized image.

BEFORE YOU BEGIN

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

On workstation, run the **lab customization-review setup** command. This ensures that the required packages are installed on workstation, and provisions the environment with the resources needed to launch an instance.

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

1. On workstation, retrieve the **osp-small.qcow2** image from http://materials.example.com/osp-small.qcow2 and save it in the **/home/student/** directory.
2. Create a copy of the **/usr/share/diskimage-builder/elements** directory to work with under **/home/student/**.
3. Create a **post-install.d** directory under the working copy of the **rhel7** directory.
4. Add a script under the **rhel7/post-install.d** directory to enable the httpd service.
5. Export the following environment variables, which diskimage-builder requires. The DIB_NO_TMPFS variable is only required because workstation has a small hard disk, it is not required in most cases.

Environment Variables

VARIABLE	CONTENT
DIB_LOCAL_IMAGE	**/home/student/osp-small.qcow2**
DIB_YUM_REPO_CONF	**"/etc/yum.repos.d/openstack.repo"**

VARIABLE	CONTENT
ELEMENTS_PATH	**/home/student/elements**
DIB_NO_TMPFS	**1**

6. Build a Red Hat Enterprise Linux 7 image named **production-rhel-web.qcow2** using the diskimage-builder elements configured previously. Include the *httpd* package in the image. Provide the password for `student` when prompted.
7. Use the **guestfish** command to add a custom web index page to the **production-rhel-web.qcow2** image. Include the text **production-rhel-web** in the **index.html** file. Ensure the SELinux context of **/var/www/html/index.html** is correct.
8. As the `operator1` user, create a new OpenStack image named `production-rhel-web` using the **production-rhel-web.qcow2** image, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.
9. As the `operator1` user, launch an instance using the following attributes:

   Instance Attributes

ATTRIBUTE	VALUE
flavor	**default**
key pair	**example-keypair**
network	**production-network1**
image	**production-rhel-web**
security group	**default**
name	**production-web1**

10. List the available floating IP addresses, and then allocate one to `production-web1`.
11. Use the **ssh** command to log in to the `production-web1` instance. Verify that the *httpd* package is installed, and that the `httpd` service is enabled and running.
12. From `workstation`, confirm that the custom web page, displayed from `production-web1`, contains the text **production-rhel-web**.

Evaluation

On `workstation`, run the **lab customization-review grade** command to confirm the success of this exercise. Correct any reported failures and rerun the command until successful.

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

Cleanup

On `workstation`, run the **lab customization-review cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab customization-review cleanup
```

PERFORMING IMAGE OPERATIONS

PERFORMANCE CHECKLIST

In this lab, you will build a disk image using diskimage-builder, modify it using **guestfish**, upload it to the Image service, then launch an instance using the new image.

OUTCOMES

You should be able to:

- Build an image using diskimage-builder.
- Customize the image using a post-install script.
- Customize the image using the **guestfish** command.
- Upload the image to the OpenStack Image service.
- Launch an instance using the customized image.

BEFORE YOU BEGIN

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

On workstation, run the **lab customization-review setup** command. This ensures that the required packages are installed on workstation, and provisions the environment with the resources needed to launch an instance.

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

1. On workstation, retrieve the **osp-small.qcow2** image from http://materials.example.com/osp-small.qcow2 and save it in the **/home/student/** directory.

```
[student@workstation ~]$ wget http://materials.example.com/osp-small.qcow2 \
-O /home/student/osp-small.qcow2
```

2. Create a copy of the **/usr/share/diskimage-builder/elements** directory to work with under **/home/student/**.

```
[student@workstation ~]$ cp -a /usr/share/diskimage-builder/elements \
/home/student/
```

3. Create a **post-install.d** directory under the working copy of the **rhel7** directory.

```
[student@workstation ~]$ mkdir -p /home/student/elements/rhel7/post-install.d
```

4. Add a script under the **rhel7/post-install.d** directory to enable the httpd service.
   4.1. Add a script to enable the httpd service.

```
[student@workstation ~]$ cd /home/student/elements/rhel7/post-install.d/
[student@workstation post-install.d]$ cat <<EOF > 01-enable-services
#!/bin/bash
systemctl enable httpd
EOF
```

4.2. Set the executable permission on the script.

```
[student@workstation post-install.d]$ chmod +x 01-enable-services
```

4.3. Change back to your home directory.

```
[student@workstation post-install.d]$ cd
[student@workstation ~]$
```

5. Export the following environment variables, which diskimage-builder requires. The `DIB_NO_TMPFS` variable is only required because `workstation` has a small hard disk, it is not required in most cases.

Environment Variables

VARIABLE	CONTENT
DIB_LOCAL_IMAGE	**/home/student/osp-small.qcow2**
DIB_YUM_REPO_CONF	**"/etc/yum.repos.d/openstack.repo"**
ELEMENTS_PATH	**/home/student/elements**
DIB_NO_TMPFS	**1**

```
[student@workstation ~]$ export DIB_LOCAL_IMAGE=/home/student/osp-small.qcow2
[student@workstation ~]$ export DIB_YUM_REPO_CONF=/etc/yum.repos.d/openstack.repo
[student@workstation ~]$ export ELEMENTS_PATH=/home/student/elements
[student@workstation ~]$ export DIB_NO_TMPFS=1
```

6. Build a Red Hat Enterprise Linux 7 image named **`production-rhel-web.qcow2`** using the diskimage-builder elements configured previously. Include the *httpd* package in the image. Provide the password for `student` when prompted.

```
[student@workstation ~]$ disk-image-create vm rhel7 \
-t qcow2 \
-p httpd \
-o production-rhel-web.qcow2
```

7. Use the **`guestfish`** command to add a custom web index page to the **`production-rhel-web.qcow2`** image. Include the text **`production-rhel-web`** in the **`index.html`** file. Ensure the SELinux context of **`/var/www/html/index.html`** is correct.

7.1. Open a **`guestfish`** shell for the **`production-rhel-web.qcow2`** image.

```
[student@workstation ~]$ guestfish -i -a production-rhel-web.qcow2
...output omitted...
```

```
><fs>
```

7.2. Create a new **/var/www/html/index.html** file.

```
><fs> touch /var/www/html/index.html
```

7.3. Edit the **/var/www/html/index.html** file and include the required key words.

```
><fs> edit /var/www/html/index.html
This instance uses the production-rhel-web image.
```

7.4. To ensure the new index page works with SELinux in enforcing mode, restore the **/var/www/** directory context (including the **index.html** file).

```
><fs> selinux-relabel /etc/selinux/targeted/contexts/files/file_contexts /var/www/
```

7.5. Exit the **guestfish** shell.

```
><fs> exit

[student@workstation ~]$
```

8. As the `operator1` user, create a new OpenStack image named `production-rhel-web` using the **production-rhel-web.qcow2** image, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.

8.1. Source the **/home/student/operator1-production-rc** environment file.

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

8.2. Upload the **production-rhel-web.qcow2** image to the OpenStack Image service.

```
[student@workstation ~(operator1-production)]$ openstack image create \
--disk-format qcow2 \
--min-disk 10 \
--min-ram 2048 \
--file ~/production-rhel-web.qcow2 \
production-rhel-web
...output omitted...
```

9. As the `operator1` user, launch an instance using the following attributes:

Instance Attributes

ATTRIBUTE	VALUE
flavor	**default**
key pair	**example-keypair**
network	**production-network1**
image	**production-rhel-web**

ATTRIBUTE	VALUE
security group	**default**
name	**production-web1**

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=production-network1 \
--image production-rhel-web \
--security-group default \
--wait production-web1
...output omitted...
```

10. List the available floating IP addresses, and then allocate one to `production-web1`.

10.1. List the floating IPs. Available IP addresses have the Port attribute set to **None**.

```
[student@workstation ~(operator1-production)]$ openstack floating ip list \
-c "Floating IP Address" -c Port
+---------------------+------+
| Floating IP Address | Port |
+---------------------+------+
| 172.25.250.104      | None |
+---------------------+------+
```

10.2. Attach an available floating IP to the `production-web1` instance.

```
[student@workstation ~(operator1-production)]$ openstack server add \
floating ip production-web1 172.25.250.104
```

11. Use the **ssh** command to log in to the `production-web1` instance. Verify that the *httpd* package is installed, and that the `httpd` service is enabled and running.

11.1. Use SSH to log in to the `production-web1` instance.

```
[student@workstation ~(operator1-production)]$ ssh cloud-user@172.25.250.104
Warning: Permanently added '172.25.250.104' (ECDSA) to the list of known hosts.
[cloud-user@production-web1 ~]$
```

11.2. Verify that the *httpd* package is installed.

```
[cloud-user@production-web1 ~]$ rpm -q httpd
httpd-2.4.6-45.el7.x86_64
```

11.3. Verify that the `httpd` service is running, and then log off.

```
[cloud-user@production-web1 ~]$ systemctl status httpd
...output omitted...
   Loaded: loaded (/usr/lib/systemd/system/httpd.service; enabled; vendor preset:
 disabled)
   Active: active (running) since Wed 2017-05-24 23:55:42 EDT; 8min ago
     Docs: man:httpd(8)
           man:apachectl(8)
```

```
 Main PID: 833 (httpd)
...output omitted...
[cloud-user@production-web1 ~]$ logout
[student@workstation ~(operator1-production)]$
```

12. From `workstation`, confirm that the custom web page, displayed from `production-web1`, contains the text **`production-rhel-web`**.

```
[student@workstation ~(operator1-production)]$ curl http://172.25.250.104/
index.html
This instance uses the production-rhel-web image.
```

Evaluation

On `workstation`, run the **`lab customization-review grade`** command to confirm the success of this exercise. Correct any reported failures and rerun the command until successful.

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

Cleanup

On `workstation`, run the **`lab customization-review cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab customization-review cleanup
```

SUMMARY

In this chapter, you learned:

- The advantages and disavantages of building an image as opposed to customizing an existing one, such as meeting organization security standards, including third-party agents, and adding operator accounts.
- When to use the **`guestfish`** and **`virt-customize`** tools. Use **`guestfish`** when you need to perform low-level tasks such as partitioning disks. Use **`virt-customize`** for all common customization tasks such as setting passwords and installing packages.
- Making changes to an image using these tools affects SELinux file contexts, because SELinux is not supported directly in the **`chroot`** environment.
- To avoid image sprawl, use a smaller set of images and perform per-instance customization using **cloud-init** or a configuration management system as required.

CHAPTER 5

MANAGING STORAGE

GOAL

Manage Ceph and Swift storage for OpenStack.

OBJECTIVES

- Explain persistent storage options for use in OpenStack, focusing on the expanding capabilities of the default Ceph storage.
- Discuss the Swift and Ceph object stores, comparing architectural considerations.
- Describe the methods for configuring the new shared file systems component.
- Interpreting the behavior of ephemeral storage configuration choices.

SECTIONS

- Implementing Block Storage (and Guided Exercise)
- Comparing Object Stores (and Guided Exercise)
- Managing Shared File Systems (and Guided Exercise)
- Managing Ephemeral and Persistent Storage (and Guided Exercise)

LAB

- Managing Storage

IMPLEMENTING BLOCK STORAGE

OBJECTIVES

After completing this section, students should be able to describe back-end storage options for OpenStack services.

STORAGE IN RED HAT OPENSTACK PLATFORM

Red Hat OpenStack Platform supports a wide range of storage options, from simple back ends and direct-attached storage (DAS) up to the largest enterprise storage area network (SAN) and network-attached storage (NAS) devices and hardware. Administrators can choose solutions based on cost, performance, scalability, expandability, existing infrastructure reuse, and ease of setup for configurations ranging from the smallest proof of concept configuration to massive distributed clusters.

Red Hat OpenStack Platform provides three types of persistent storage: block storage, object storage, and file-based storage. Block storage is discussed in this section, object and file-based storage follow in later sections. Block storage supplies block devices which can be structured for use in many ways, identical to practices learned in legacy enterprise computing. Block devices can be used to build mountable filesystems, NFS-shared volumes, raw database device files, and for use as bootable system disks.

Compute nodes can be configured with local disks (DAS) to be used for block storage, in specific use cases. Typically, local compute node disks are not recommend for most cloud use cases because they do not easily support requisite cloud concepts such as scalability, redundancy, and the ability to perform live migrations without block-based copying between compute nodes. However, there are significant cases for direct-attached compute node storage, especially using SSDs. One example is cloud-based databases, in which the deployed application is made redundant and resilient not by connecting instances to shared storage, but by clustering the database across SSDs attached to each of multiple instances.

NOTE

It is a common misconception among new OpenStack administrators that the concept of the compute service's ephemeral storage is equivalent to local compute node direct-attached disk storage. In early versions of OpenStack, local disk storage was the only configuration supporting ephemeral storage, but now almost any block storage choice can support ephemeral virtual devices, which are destroyed when the instance is terminated. An explanation and comparison of ephemeral and persistent storage concepts are found in an upcoming section in this chapter.

BLOCK STORAGE CHOICES

Block storage is implemented in OpenStack by the block storage service (cinder), allowing users to access block storage devices as volumes. Volumes are persistent, meaning that they can be attached, detached and reattached to one or another instance while the stored data remains intact.

The block storage service uses block storage drivers to support different back ends, each of which is designed to support a shared community or vendor-specific storage solution. Many enterprise hardware solutions are supported by their vendors, allowing instances to gain direct access to the

back-end's physical block devices to obtain the highest performance in read/write IO. However, support for utilizing files as volumes is now common, using NFS, GlusterFS and CephFS.

Red Hat OpenStack Platform supports several commodity storage back end technologies, which can be implemented in many different combinations.

LVM and iSCSI

The Linux-based Logical Volume Manager (LVM) exposes local physical disks as logical volumes to the operating system. The LVM back-end implements block storage as LVM logical partitions. Each compute host implement LVM must have a volume group dedicated to the block storage service. Although LVM is the default back end for the block storage service, Red Hat does not support LVM for production environments, recommending its use for test environments only. Remote LVM devices are typically transported using the iSCSI networking protocol. Although network connectivity can be achieved at a lower cost compared to other storage back end technologies since iSCSI does not require host bus adaptors (HBA) or storage-specific network devices Red Hat does not support ISCSI for production environments.

Red Hat Ceph Storage

Red Hat Ceph Storage provides petabyte-scale storage and can be configured in a highly resilient and redundant architecture. Ceph provides support for the object storage API as the object service (swift), and can be used as a back end for the block storage service (cinder) and the image service (glance). Ceph also supports accessing volumes through NFS, and the distributed file system interface CephFS.

NFS

Using the Network File System (NFS) file system protocol, file clients can mount remote file systems and access contents using Remote Procedure Calls (RPC), with files being used as block-device volumes. NFS has a cost advantage by using shared NICs and traditional networking. Although simpler, NFS may not exhibit the best throughput or lowest latency when compared to other cloud-design block service back ends such as Ceph or Gluster.

Vendor-specific Storage

Supported storage hardware vendors provide drivers for Red Hat OpenStack Platform services to use their storage infrastructure as a back end. Red Hat does not provide support for vendor-specific storage; the partner storage provider supports their offering directly.

THE RED HAT CEPH STORAGE ARCHITECTURE

Red Hat Ceph Storage is designed to provide an enterprise-ready, software-defined storage solution that scales out to multipetabytes, using servers based on industry-standard hardware and storage devices.

Ceph is based on a modular and distributed architecture that contains the following elements:

- An object storage back end known as RADOS (Reliable Autonomic Distributed Object Store)
- A variety of access methods to interact with RADOS

RADOS is a self-healing and self-managing software-based object store.

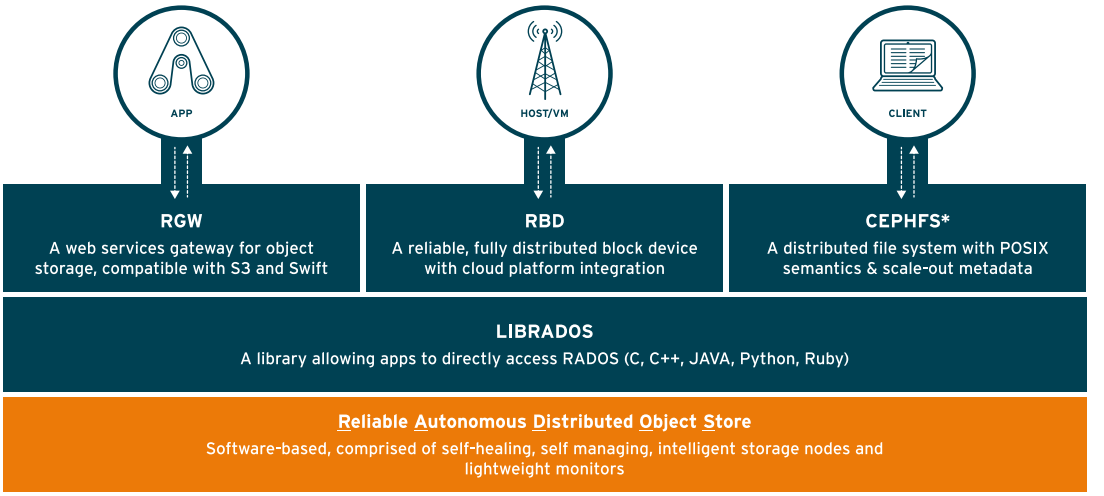


Figure 5.1: Ceph Cluster RADOS back end

CEPH TERMINOLOGY

Ceph cluster

A group of nodes to ensure continued operability of the Red Hat Ceph Storage system. The `/etc/ceph/ceph.conf` configuration file defines the cluster settings.

Node

A *node* can be any machine with any of the Ceph components active in it. The are multiple types of node, such as administration nodes, monitor nodes, and storage nodes. The role of the node depends on the components that are active.

Pool

Pools are logical partitions of the Ceph storage cluster, used to store objects under a common name tag. Each pool is assigned a specific number of hash buckets to group objects together for storage. These hash buckets are called *Placement Groups (PGs)*.

Placement Groups

A Placement Group (PG) aggregates a series of objects into a hash bucket, or group, and is mapped to a set of *Object Storage Devices (OSDs)*. An object belongs to only one PG, and all objects belonging to the same PG return the same hash result.

CEPH STORAGE BACK END COMPONENTS

RADOS, the Ceph storage back end, is based on the following daemons, which can be scaled out to meet the requirements of the architecture being deployed:

- *Monitors (MONs)*, which maintain maps of the cluster state and are used to help the other daemons coordinate with each other.
- *Object Storage Devices (OSDs)*, which store data and handle data replication, recovery, and rebalancing.
- *Managers (MGRs)*, which keep track of runtime metrics and expose cluster information through a web UI dashboard and REST API.
- *Metadata Servers (MDSs)*, which store metadata used by CephFS (but not object storage or block storage) to allow efficient POSIX command execution by clients.

The following figure shows the distribution of these components on overcloud nodes using common roles.

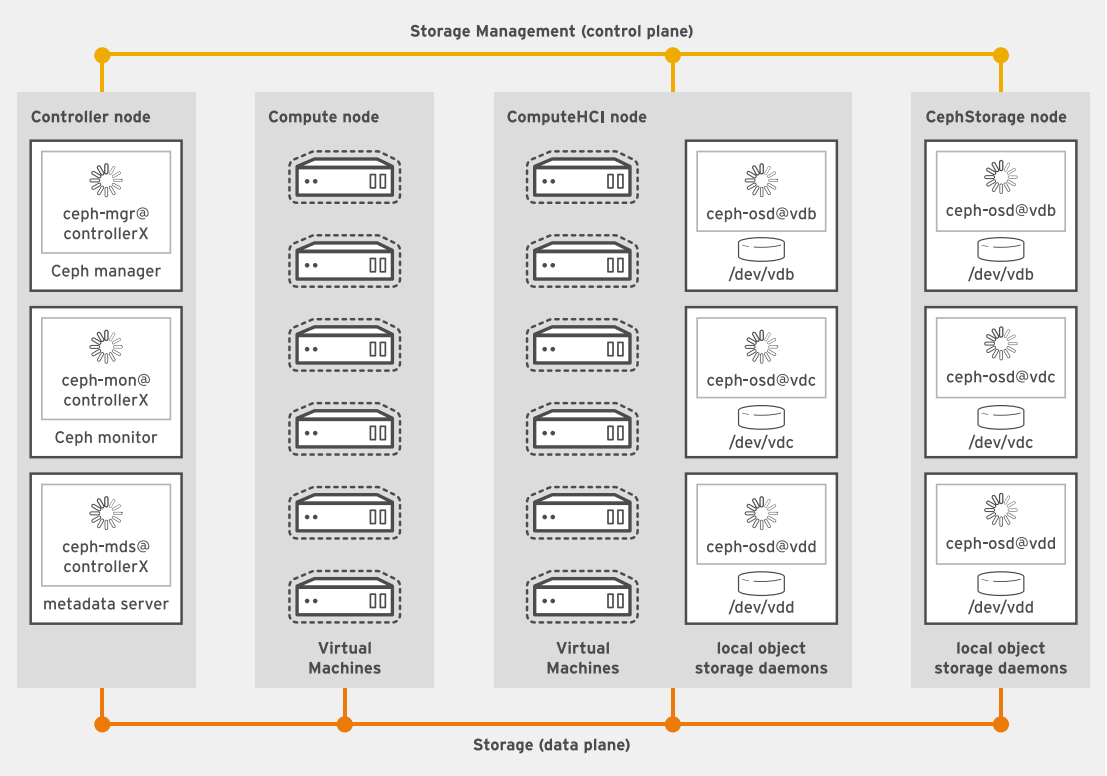


Figure 5.2: Ceph component placement on overcloud nodes

AUTHENTICATION WITH CEPHX

In a Red Hat Ceph Storage cluster, *users* are used to authorize communication between clients, applications, and daemons in the cluster. To securely authenticate these users, Ceph uses the `cephx` authentication protocol, based on shared secret keys.

The installation process enables `cephx` by default, so that the Red Hat Ceph Storage cluster requires authentication and proper authorization by all client applications.

Ceph uses user accounts for several purposes:

- For internal communication between Ceph daemons
- For client applications accessing the storage cluster through the `librados` library
- For Ceph administrators

Accounts used by Ceph daemons have names that match the associated daemon: `osd.1` or `mgr.serverc`. The Ansible Playbook creates them during installation.

Accounts used by client applications using `librados` have names that start with `client.`. For example, when integrating OpenStack with ceph, it is common to create the dedicated `client.openstack` user account. Also, when Ansible deploys the Ceph Object Gateway, it creates the dedicated account called `client.rgw.`*`hostname`*. Finally, if you are developing custom software on top of `librados`, you should also create a specific account for it with appropriate capabilities.

Accounts used by administrators also have names that start with `client.`, which are used when running commands such as **`ceph`** and **`rados`**. The installer creates the superuser account `client.admin`, with capabilities that allow it to access everything and to modify the cluster configuration. Ceph uses `client.admin` by default when you run administrative commands without explicitly specifying a username with the **`--name`** or **`--id`** options.

End users of a ceph-aware application do not have an account on the Red Hat Ceph Storage cluster. Rather, they access the application, which then accesses Ceph on their behalf.

From the Ceph point of view, the application is the client. The application may provide its own end-user authentication through other mechanisms.

The Key-ring File

For authentication, clients must be configured with a cephx username and a key-ring file containing the user's secret key. A cephx user needs this key-ring file to access the Red Hat Ceph Storage cluster.

Ceph automatically generates the key-ring file for each user account you create. However, you still need to copy this file to the client systems or application servers on which it is needed.

On these client systems, `librados` uses the **`keyring`** parameter from the **`/etc/ceph/ceph.conf`** configuration file to locate the key-ring file. Its default value is **`/etc/ceph/$cluster.$name.keyring`**. For example, for the `client.openstack` account, the key-ring file is **`/etc/ceph/ceph.client.openstack.keyring`**.

The key-ring file stores the secret key as plain text. It must be protected with appropriate Linux file permissions so that it can only be read by Linux users that need access. Only deploy the key-ring file for a cephx user on systems that need it for authentication.

NOTE

The cephx protocol does not *transmit* the shared secret keys as plain text. Instead, a client requests a session key for its user from a Monitor, which provides it to the client encrypted with the client's shared secret key. If a client knows its secret key, it can decrypt the session key and use the session key to request credentials from a Monitor to authenticate to cluster daemons. This is similar to how the Kerberos protocol works, with a cephx key-ring file being comparable to a Kerberos keytab file.

A more detailed discussion of the protocol is available from the upstream Ceph project's documentation at *High Availability Authentication* [http://docs.ceph.com/docs/master/architecture/#high-availability-authentication].

Command-line Tools

With command-line tools such as **`ceph`**, **`rados`**, and **`rbd`**, administrators can specify the user account and the key-ring file to use with the **`--id`** and **`--keyring`** options. Without them, the commands authenticate as the `client.admin` user.

For example, the following **`ceph`** command authenticates as `client.operator3` to list the available pools:

```
[root@controller ~]# ceph --id operator3 osd lspools
1 myfirstpool,2 mysecondpool,
```

IMPORTANT

You must not include the `client.` prefix when using the **`--id`** option. The **`--id`** option automatically assumes that prefix is being used. As an alternative, you may use the **`--name`** option, which requires the `client.` prefix.

If you store the key-ring file in its default location, you do not need the **`--keyring`** option.

AUTHORIZATION WITH CEPHX

When you create a new user account, you need to grant it permissions to control what that user is authorized to do in the Red Hat Ceph Storage cluster. Permissions within `cephx` are known as *capabilities*, and you grant them by daemon type (`mon`, `osd`, `mgr`, or `mds`).

You can use capabilities to restrict access to data in a pool, a pool's namespace, or a set of pools based on application tags. They also allow the daemons in the Red Hat Ceph Storage cluster to interact with each other.

Cephx Capabilities

Within `cephx`, and for each daemon type, several capabilities are available:

- **`r`** grants read access. Each user account should have at least read access on the Monitors to be able to retrieve the CRUSH map.
- **`w`** grants write access. Clients need write access to store and modify objects on OSDs. For Managers (MGRs), **`w`** grants the right to enable or disable modules.
- **`x`** grants authorization to execute extended object classes. This allows clients to perform extra operations on objects such as setting locks with **`rados lock get`** or listing RBD images with **`rbd list`**.
- **`*`** grants full access.
- `class-read` and `class-write` are subsets of `x`. You mostly use them on pools used for RBD.

USER MANAGEMENT

Listing Users and Their Capabilities

To list existing user accounts, use the **`ceph auth list`** command:

```
[root@controller ~]# ceph auth list
...output omitted...
osd.0
 key: AQBW6Tha5z6OIhAAMQ7nY/4MogYecxKqQxX1sA==
 caps: [mgr] allow profile osd
 caps: [mon] allow profile osd
 caps: [osd] allow *
client.admin
 key: AQCi6Dhajw7pIRAA/ECkwyipx2/raLWjgbklyA==
 caps: [mds] allow *
 caps: [mgr] allow *
 caps: [mon] allow *
 caps: [osd] allow *
...output omitted...
```

To get the details of a specific account, use the **`ceph auth get`** command:

```
[root@controller ~]# ceph auth get client.admin
exported keyring for client.admin
[client.admin]
 key = AQCi6Dhajw7pIRAA/ECkwyipx2/raLWjgbklyA==
 caps mds = "allow *"
 caps mgr = "allow *"
 caps mon = "allow *"
 caps osd = "allow *"
```

You can also just print the secret key as follows:

```
[root@controller ~]# ceph auth print-key client.admin
AQCi6Dhajw7pIRAA/ECkwyipx2/raLWjgbklyA==
```

Creating New User Accounts

The **`ceph auth get-or-create`** command creates a new user account and generates its secret key.

The command prints this key to **`stdout`** by default, so it is common to redirect the command output with the greater-than symbol (>) to save it in a key-ring file.

The following example creates the `application1` user account with read and write access to all pools, and stores the key-ring file as **`/etc/ceph/ceph.client.application1.keyring`**. If you omit the capabilities the user will not have any rights to any pool. They can still authenticate but not get access to any pool. You can add the required capabilities later as required.

```
[root@controller ~]# ceph auth get-or-create client.application1 \
mon 'allow r' \
osd 'allow rw' \
> /etc/ceph/ceph.client.application1.keyring
```

Remember that authentication requires the key-ring file so you must copy it to all client systems that use this new user account.

OPENSTACK IMAGE SERVICE WITH RED HAT CEPH STORAGE

The following table outlines the parameters that the image service uses to leverage Red Hat Ceph Storage as the back-end storage provider for storing its images. These parameters appear under the **`[glance_store]`** INI section in the configuration file (**`/etc/glance/glance-api.conf`**).

PARAMETERS	PURPOSE
`default_store`	The default object store is initially set to **`file`** to store images on the local file system. Set it to **`rbd`** to use Ceph as the image back-end storage provider.
`stores`	Lists the storage back ends that the image service knows.
`rbd_store_user`	Specifies the cephx user for cluster authentication.
`rbd_store_pool`	Sets the Ceph pool name to accommodate the instance images.
`rbd_store_ceph_conf`	Sets the path to the Ceph configuration file.
`rbd_store_chunk_size`	Images are processed in pieces of this size in megabytes. The default value is 8.

OPENSTACK BLOCK STORAGE SERVICE WITH RED HAT CEPH STORAGE

The following table outlines the parameters that assist the Block Storage service to use Red Hat Ceph Storage as the back-end storage provider for storing its volumes. The global parameters

appear under the **[DEFAULT]** INI section in the configuration file (**/etc/cinder/cinder.conf**). The back-end specific parameters appear under an INI section that has the same name as the specified back end.

PARAMETERS	PURPOSE
enabled_backends	A global parameter that enables one or more storage back ends for the Block Storage service. The value of this parameter should match that of the **volume_backend_name** parameter. You can specify a comma-separated list of back-end names.
volume_backend_name	Binds an arbitrary name with a specific driver of the storage system. The same name is used throughout the configuration file to address the storage back end.
volume_driver	Sets the storage system driver.
rbd_ceph_conf	Sets the path to the Ceph configuration file.
rbd_user	Sets the cephx user to authenticate to the cluster as.
rbd_pool	Sets the name of the Ceph pool to accommodate the volumes.

VIEWING THE CEPH CLUSTER STATUS

This section describes how to view various aspects of the status of the Ceph cluster.

Viewing the Status of the Administrative Components

Run the following command on the admin node to view the status of the administrative components of the Ceph cluster.

```
[root@controller ~]# systemctl list-units -t service ceph\*
UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
ceph-mds@controller.service loaded active running Ceph MDS
ceph-mgr@controller.service loaded active running Ceph Manager
ceph-mon@controller.service loaded active running Ceph Monitor

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

3 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

The preceding output shows that the `Ceph Manager` and the `Ceph Monitor` services are running. The `Ceph Manager` service provides for the collection of cluster statistics. In earlier versions of Ceph, most of these statistics were collected and maintained by the monitors, which caused them to become overloaded in larger cluster deployments. To remedy this problem, the collection of cluster statistics is now handled by the Ceph Manager. The Manager daemon centralizes access to all data collected from the cluster and can provide a simple web dashboard to storage administrators on TCP port 7000 (by default).

The `Ceph Monitor` service maintains a master copy of the cluster map (record). The cluster map gives details regarding the state of the cluster and its configuration. The majority of the Ceph monitor nodes should agree in case of an update to the cluster map to establish a consensus regarding the state of the cluster. To ensure a clear majority when monitors vote on the state of

the cluster, the cluster must be configured with an odd number of monitors. This also means that more than half of the configured monitors must be functional for the Ceph storage cluster to be operational and accessible. If the number of active monitors falls below this threshold, the entire cluster will become inaccessible to all clients. This is necessary to protect the integrity of the cluster's data.

Viewing the Status of the Storage Component

Run the following command on the storage node to view the status of the storage component of the Ceph cluster.

```
[root@ceph ~]# systemctl list-units -t service ceph\*
UNIT                 LOAD   ACTIVE SUB     DESCRIPTION
ceph-osd@vdb.service loaded active running Ceph OSD
ceph-osd@vdc.service loaded active running Ceph OSD
ceph-osd@vdd.service loaded active running Ceph OSD

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

3 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

The preceding output shows that there are three running instances of Object Storage Devices (OSDs) for each of the `vdb`, `vdc`, and `vdd` storage devices. Ceph Object Storage Devices (OSDs) are the building blocks of a Ceph storage cluster. OSDs connect storage devices (such as a hard disk or other block device) to the Ceph storage cluster. Each storage device is formatted with a standard file system type such as XFS. An individual storage server may run multiple OSD daemons on the basis of multiple devices and provide multiple OSDs to the cluster.

Viewing the Status of the Ceph Cluster

Run the following command on the admin node to view the status of the Ceph cluster.

```
[root@controller ~]# ceph -s
  cluster:
    id:     817893a8-b2f5-11e8-8087-52540001fac8
    health: HEALTH_OK

  services:
    mon: 1 daemons, quorum controller
    mgr: controller(active)
    mds: cephfs-1/1/1 up  {0=controller=up:active}
    osd: 6 osds: 6 up, 6 in

  data:
    pools:   7 pools, 416 pgs
    objects: 748 objects, 5655 MB
    usage:   6308 MB used, 107 GB / 113 GB avail
    pgs:     416 active+clean
```

Notice that the cluster has the unique identifier **`817893a8-b2f5-11e8-8087-52540001fac8`**. The health status **`HEALTH_OK`** represents the healthy state of the Ceph cluster. The other possible values are **`HEALTH_WARN`** and **`HEALTH_ERR`**. The **`HEALTH_WARN`** state might indicate that one of the OSDs in the cluster has failed. However, this does not mean that the entire cluster is faulty.

When a healthy OSD replaces the faulty one, the status returns to **HEALTH_OK** when the next health check is performed. The **HEALTH_ERR** state indicates that the Ceph cluster is faulty. In this state, the cluster does not allow any further I/O operations. An administrator must intervene and diagnose the cause of the error. Network disruption, the cluster reaching its maximum capacity, and other issues could result in this state.

The preceding output shows that there is only one monitor node (`controller`) in the cluster. It therefore elected itself as the leading monitor node of the cluster. In ideal situations, you will see a minimum of three monitor nodes and the majority of them agree to choose a particular monitor node as the leading monitor node of the cluster. Notice that the preceding output mentions the `Ceph Manager` service being active in `controller`. The `Ceph Metadata` service is in the active state in `controller`. Note also that there are three OSD instances signifying that the cluster is aware of three storage devices and all of them are active.

IMPORTANT

Red Hat Ceph Storage 3 only supports one active MDS at a time. All other deployed MDSes must be in standby mode to provide high availability. If the active MDS fails, one of the standby MDSes takes over. Having multiple MDSes active at the same time is an experimental feature and not recommended for production.

To verify the number of active MDS daemons, use the **`ceph fs status`** command. The following command verifies the number of active MDS daemons for a CephFS file system named `cephfs`:

```
[root@controller ~]# ceph fs status cephfs
```

The output of the **`ceph -s`** command shown in Viewing the Status of the Ceph Cluster illustrates that there are 7 pools in the cluster with 416 placement groups. The clean state of the placement groups contributes to the healthy status of the cluster. Notice the number of objects and the space they consume. You may also notice the usage statistics of the Ceph cluster.

MANAGING CEPH-BACKED PERSISTENT VOLUMES

This section describes the management of persistent volumes backed by Ceph.

Checking the Health and Configuration Settings

If you want to use the Red Hat Ceph Storage system as the back-end storage provider for the Block Storage service, you should first check the health of the Ceph cluster. If the Ceph cluster is healthy, you should then verify the configuration settings of the Block Storage service. The configuration settings of the Block Storage service should look similar to the following:

```
[DEFAULT]
...output omitted...
enabled_backends=tripleo_ceph
...output omitted...
[tripleo_ceph]
volume_backend_name=tripleo_ceph
volume_driver=cinder.volume.drivers.rbd.RBDDriver
rbd_ceph_conf=/etc/ceph/ceph.conf
rbd_user=openstack
rbd_pool=volumes
```

If you have multiple back ends active for the Block Storage service at the same time, then you need to create a volume type and bind it to the appropriate storage back end. The tenant users can bind a volume type with a specific storage back end by setting the **`volume_backend_name`** key to the same value as in the Block Storage service configuration settings for that back end. Before you set the **`volume_backend_name`** attribute, inspect the settings of the Block Storage service to determine the value of the parameter. This way of binding a volume type with a storage back end allows the tenant users to select the volume type while creating the volume to use a specific storage back end. To create a volume type, run the following command as an administrative user.

```
[user@demo ~]$ openstack volume type create \
--property volume_backend_name=tripleo_ceph cephvoltype
+-------------+--------------------------------------+
| Field       | Value                                |
+-------------+--------------------------------------+
| description | None                                 |
| id          | 0236db8a-a06f-441b-970d-3ec8fe57f5bd |
| is_public   | True                                 |
| name        | cephvoltype                          |
| properties  | volume_backend_name='tripleo_ceph'   |
+-------------+--------------------------------------+
```

Run the following command to create an empty persistent volume of `cephvoltype` volume type.

```
[user@demo ~]$ openstack volume create \
--size 1 --type cephvoltype cephvol1
...output omitted...
```

Run the following command to verify the availability of the volume.

```
[user@demo ~]$ openstack volume list
+-------------+----------+-----------+------+-------------+
| ID          | Name     | Status    | Size | Attached to |
+-------------+----------+-----------+------+-------------+
| f2f7...b575 | cephvol1 | available |    1 |             |
+-------------+----------+-----------+------+-------------+
```

Confirm that an object exists in the Ceph pool with the matching unique identifier as the volume.

```
[root@controller ~]# rados -p volumes ls
rbd_id.volume-f2f7...b575
rbd_directory
rbd_info
rbd_object_map.fbba63e2746d
rbd_header.fbba63e2746d
```

Run the following command to create another persistent volume to act as the root file system for an instance using the existing `rhel7` image.

```
[user@demo ~]$ openstack volume create \
--size 10 --type cephvoltype --image rhel7 cephvol2
...output omitted...
```

IMPORTANT
The volume size depends on the image request. Always review the image disk requirements before creating the volume. Use the **`openstack image show <image_name>`** command to view image details.

Verify that a new object is created in the Ceph pool to represent the new volume.

```
[root@controller ~]# rados -p volumes ls | grep volume
rbd_id.volume-f2f7eecc-4649-4761-a1fd-3467782eb575
rbd_id.volume-d66b4577-3162-4744-a3d0-d97d77d93d2f
```

INTRODUCING CINDER MULTI-ATTACH

Red Hat OpenStack Platform supports attaching persistent block storage volumes to more than one instance at the same time, using either read-write or read-only attachments. Volume multi-attach allows multiple instances to connect to a volume, providing for faster failover to a standby instance. Clustered applications with two nodes where one is active and one is passive. Both require access to the same volume although only one accesses actively. When the active one goes down, the passive one can take over quickly and has access to the data.

Although block devices can be configured for multi-attach access, file systems on top of those volumes can be mounted to only one system at a time to avoid file system corruption. During a failover between instances, the file sytem must be unmounted on the failed instance before being mounted to the second instance. Use of legacy solutions such as the GFS clustered file system is not supported in OpenStack.

Creating a Multiattach-capable Volume

Creating a multiattach-capable volume involves three steps.

- As an administrative user, create a volume type.

```
[user@demo ~]$ cinder type-create volume-multi
```

- As an administrative user, set the volume type **`multiattach`** key to **`"<is> True"`**.

```
[user@demo ~]$ cinder type-key volume-multi set multiattach="<is> True"
```

- Create a volume for the defined type.

```
[user@demo ~]$ cinder create 2 --name multi-volume1 --volume-type volume-multi
```

Current characteristics of the multi-attach volume feature include:

- You cannot change the type of a multi-attach volume while it's state is **`in-use`**. The volume state must be **`available`** before you can change its type.
- Red Hat Ceph Storage does not currently support the multi-attach feature, but it is in development. Other block storage back ends already support multi-attach.
- An attach mode is not yet available for secondary volume attachments. By default, all secondary attachments are read/write mode, which could cause volume migration issues.

REFERENCES

Further information is available in the *Storage Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en/red-hat-openstack-platform

REFERENCES

Further information about *Storage Formats* is available at
File storage, block storage, or object storage
https://www.redhat.com/en/topics/data-storage/file-block-object-storage

Further information about *Ceph Architecture* is available at
http://docs.ceph.com/docs/mimic/architecture/

Further information about *User Management* is available at
https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/html-single/administration_guide/#user_management

Further information about *Volume multi-attach* is available at
https://docs.openstack.org/cinder/latest/admin/blockstorage-volume-multiattach.html

IMPLEMENTING BLOCK STORAGE

In this exercise, you will verify the health of a Ceph cluster. You will also verify and test the settings to use the Ceph cluster as the back end for the OpenStack image service and the OpenStack Block Storage service.

OUTCOMES

You should be able to:

- Check Ceph health and configuration (from controller).
- Create an image and locate it in Ceph.
- Create a volume and locate it in Ceph.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab storage-backend setup`** command. This ensures that the appropriate flavor, network, subnet, security group, and the user's environment file exist to support the steps involved in this guided exercise.

```
[student@workstation ~]$ lab storage-backend setup
```

1. From `controller0` as `root`, verify that the `ceph-mon` service is active and running.

   1.1. Open an SSH session to `controller0` as `root`.

```
[student@workstation ~]$ ssh root@controller0
```

   1.2. Use the **`systemctl`** command to view the status of the `ceph-mon` service.

```
[root@controller0 ~]# systemctl list-units -t service ceph\*
UNIT                          LOAD   ACTIVE SUB     DESCRIPTION
ceph-mds@controller0.service loaded active running Ceph MDS
ceph-mgr@controller0.service loaded active running Ceph Manager
ceph-mon@controller0.service loaded active running Ceph Monitor

LOAD   = Reflects whether the unit definition was properly loaded.
ACTIVE = The high-level unit activation state, i.e. generalization of SUB.
SUB    = The low-level unit activation state, values depend on unit type.

3 loaded units listed. Pass --all to see loaded but inactive units, too.
To show all installed unit files use 'systemctl list-unit-files'.
```

2. View the status of the Ceph cluster in the classroom environment. Run the **`ceph -s`** command to view the status of the Ceph cluster.

```
[root@controller0 ~]# ceph -s
  cluster:
    id:     817893a8-b2f5-11e8-8087-52540001fac8
    health: HEALTH_OK

  services:
    mon: 1 daemons, quorum controller0
    mgr: controller0(active)
    mds: cephfs-1/1/1 up  {0=controller0=up:active}
    osd: 3 osds: 3 up, 3 in

  data:
    pools:   7 pools, 416 pgs
    objects: 603 objects, 4498 MB
    usage:   4830 MB used, 53504 MB / 58334 MB avail
    pgs:     416 active+clean
```

Notice that the current health status of the cluster is **HEALTH_OK**. This indicates that the Ceph cluster is operational.

3. Verify that the Ceph settings are appropriately defined with the right pool name such that the OpenStack image service uses the Ceph storage as the back-end storage provider to accommodate an image.

3.1. Retrieve the Ceph settings from the configuration file of the OpenStack image service API container.

```
[root@controller0 ~]# docker \
exec -it glance_api grep -Ei 'rbd|ceph' \
/etc/glance/glance-api.conf | grep -v ^#
stores=http,rbd
default_store=rbd
rbd_store_pool=images
rbd_store_user=openstack
rbd_store_ceph_conf=/etc/ceph/ceph.conf
```

4. Verify that the required `images` Ceph pool and the cephx user account `client.openstack` are available to configure Ceph storage as the back-end storage provider for the OpenStack image service.

4.1. On `controller0`, verify that the `images` pool is available using the **`ceph osd lspools`** command.

```
[root@controller0 ~]# ceph osd lspools
1 images,2 metrics,3 backups,4 vms,5 volumes,6 manila_data,7 manila_metadata,
```

NOTE

The later steps of this guided exercise use the `volumes` pool as well.

4.2. Verify that the cephx user account `client.openstack` is available using the **`ceph auth list`** command.

```
[root@controller0 ~]# ceph auth list
```

```
...output omitted...
client.openstack
 key: AQA+CZNbAAAAABAAXpuXxzqYw3mwT7Vz2TQ8jQ==
 caps: [mds]
 caps: [mgr] allow *
 caps: [mon] profile rbd
 caps: [osd] profile rbd pool=volumes, profile rbd pool=backups, profile rbd
 pool=vms, profile rbd pool=images, profile rbd pool=metrics
...output omitted...
```

Notice that the cephx user `client.openstack` has block-based access to the Ceph storage as defined by the `rbd` capability profile for the `images` pool.

4.3. Log off from `controller0`.

```
[root@controller0 ~]# logout
[student@workstation ~]$
```

5. On `workstation`, create an OpenStack glance image as `developer1` in the `finance` project. The image file is available at `http://materials.example.com/osp-small.qcow2`.

5.1. Source the credential file of `developer1`.

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

NOTE

The credential file for `developer1` is created while setting up for the activity during the initial steps.

5.2. Download the image file from `http://materials.example.com/osp-small.qcow2`.

```
[student@workstation ~(developer1-finance)]$ wget \
http://materials.example.com/osp-small.qcow2
...output omitted...
Saving to: 'osp-small.qcow2'

100%[=====================================>] 1,528,233,984 58.2MB/s   in 26s
...output omitted...
```

5.3. Create the OpenStack glance image with the downloaded **`/home/student/osp-small.qcow2`** file and name it `finance-image`.

```
[student@workstation ~(developer1-finance)]$ openstack image create \
--disk-format qcow2 \
--file osp-small.qcow2 \
finance-image
...output omitted...
```

5.4. Verify that the new `finance-image` image is in an **`active`** state.

```
[student@workstation ~(developer1-finance)]$ openstack image list
```

```
+--------------------------------------+---------------+--------+
| ID                                   | Name          | Status |
+--------------------------------------+---------------+--------+
| d0712086-6594-4620-b936-e58573e9c451 | finance-image | active |
| 988c81e7-bfd7-45eb-8d51-c429be83315b | rhel7         | active |
| b76aef5f-3bc2-4297-a448-cff9b80c56cf | rhel7-db      | active |
| dbe819bc-a82e-4462-9711-8005760f0c90 | rhel7-web     | active |
+--------------------------------------+---------------+--------+
```

6. View the contents of the `images` Ceph pool to verify that the image ID from the pool matches what the OpenStack image service displays.

6.1. Open an SSH session to `controller0` as `root`.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0
[root@controller0 ~]#
```

6.2. View the contents of the `images` Ceph pool to verify the image ID.

The image ID may differ in your environment and you need to use the image ID of your environment in the preceding command.

```
[root@controller0 ~]# rados -p images ls | grep \
d0712086-6594-4620-b936-e58573e9c451
rbd_id.d0712086-6594-4620-b936-e58573e9c451
```

6.3. Log off from `controller0`.

```
[root@controller0 ~]# logout
[student@workstation ~(developer1-finance)]$
```

7. Launch a new instance as `developer1` in the `finance` project with the following attributes:

- **`Flavor`**: **`default`**
- **`Image`**: **`finance-image`**
- **`Network`**: **`finance-network1`**
- **`Security Group`**: **`default`**
- **`Server Name`**: **`finance-server1`**

7.1. Run the following command to launch the new instance.

```
[student@workstation ~(developer1-finance)]$ openstack server \
create --flavor default \
--image finance-image \
--nic net-id=finance-network1 \
--security-group default \
finance-server1
...output omitted...
```

7.2. Verify that the new instance `finance-server1` is in an **`ACTIVE`** state.

```
[student@workstation ~(developer1-finance)]$ openstack server \
list -c Name -c Status
+-----------------+--------+
```

```
| Name            | Status |
+-----------------+--------+
| finance-server1 | ACTIVE |
+-----------------+--------+
```

8. As `developer1`, create a 1 GB volume called `finance-volume1` in the `finance` project. Attach the `finance-volume1` volume to the `finance-server1` instance.

8.1. Run the following command to create the volume.

```
[student@workstation ~(developer1-finance)]$ openstack volume \
create --size 1 \
finance-volume1
...output omitted...
```

8.2. Verify that the new `finance-volume1` volume is available to use.

```
[student@workstation ~(developer1-finance)]$ openstack volume list
+-------------+-----------------+-----------+------+-------------+
| ID          | Name            | Status    | Size | Attached to |
+-------------+-----------------+-----------+------+-------------+
| 92ef...4f4f | finance-volume1 | available |    1 |             |
+-------------+-----------------+-----------+------+-------------+
```

8.3. Attach `finance-volume1` to `finance-server1`.

```
[student@workstation ~(developer1-finance)]$ openstack server \
add volume \
finance-server1 finance-volume1
```

8.4. Open the console URL of the `finance-server1` instance in a browser. Log in as `root` with `redhat` as the password in the console of the instance and confirm that the new volume appears among the disks of the system. You can use the **`lsblk`** command to view the disk.

NOTE

Use the **`openstack console url show`** command to get the console URL of the instance.

```
[student@workstation ~(developer1-finance)]$ openstack console url show \
finance-server1 -f json
{
  "url": "http://172.25.250.50:6080/vnc_auto.html?token=0081acd7-
be8b-4eef-83a9-3c79a47adbc7",
  "type": "novnc"
}
```

9. Verify that the volume `finance-volume1` exists in the `volumes` Ceph pool as an object.

9.1. On `workstation`, open an SSH session to `controller0` as `root`.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0
```

```
[root@controller0 ~]#
```

9.2. Use the **rados** command to view the contents of the `volumes` pool.

```
[root@controller0 ~]# rados -p volumes ls
rbd_header.101fe33e645f4
rbd_object_map.101fe33e645f4
rbd_directory
rbd_info
rbd_id.volume-92ef6080-60ec-4017-b5ca-5e44485f4f4f
```

Confirm that the volume ID from the output of the preceding command matches what the OpenStack Block Storage service displays in the output of the **openstack volume list** command as in step 8.2. The volume ID may differ in your environment.

10. Verify that the Ceph settings are appropriately defined with the right pool name such that the OpenStack Block Storage service uses the Ceph storage as the back-end storage provider to accommodate a volume.

10.1. Run the following command to retrieve the Ceph settings from the configuration file of the OpenStack Block Storage service API container.

```
[root@controller0 ~]# docker \
exec -it cinder_api \
grep -Ei 'rbd|ceph' \
/etc/cinder/cinder.conf | grep -v ^#
enabled_backends=tripleo_ceph
[tripleo_ceph]
volume_backend_name=tripleo_ceph
volume_driver=cinder.volume.drivers.rbd.RBDDriver
rbd_ceph_conf=/etc/ceph/ceph.conf
rbd_user=openstack
rbd_pool=volumes
```

10.2. Log off from `controller0`.

```
[root@controller0 ~]# logout
[student@workstation ~(developer1-finance)]$
```

Cleanup

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

```
[student@workstation ~]$ lab storage-backend cleanup
```

This concludes the guided exercise.

COMPARING OBJECT STORES

OBJECTIVES

After completing this section, students should be able to discuss the Swift and Ceph object stores, comparing architectural considerations.

THE OPENSTACK OBJECT STORE ARCHITECTURE

The OpenStack object store service architecture has a front-end service, the proxy server (`swift_proxy`), which acts as the gateway to the rest of the components of the OpenStack object store. The proxy server maintains the object store API. The other back-end services are the account server (`swift_account_server`), container server (`swift_container_server`), and object server (`swift_object_server`) used to perform the actual storage operations in the object store system. There are additional scheduled housekeeping processes to ensure appropriate data replication, updating and error correction.

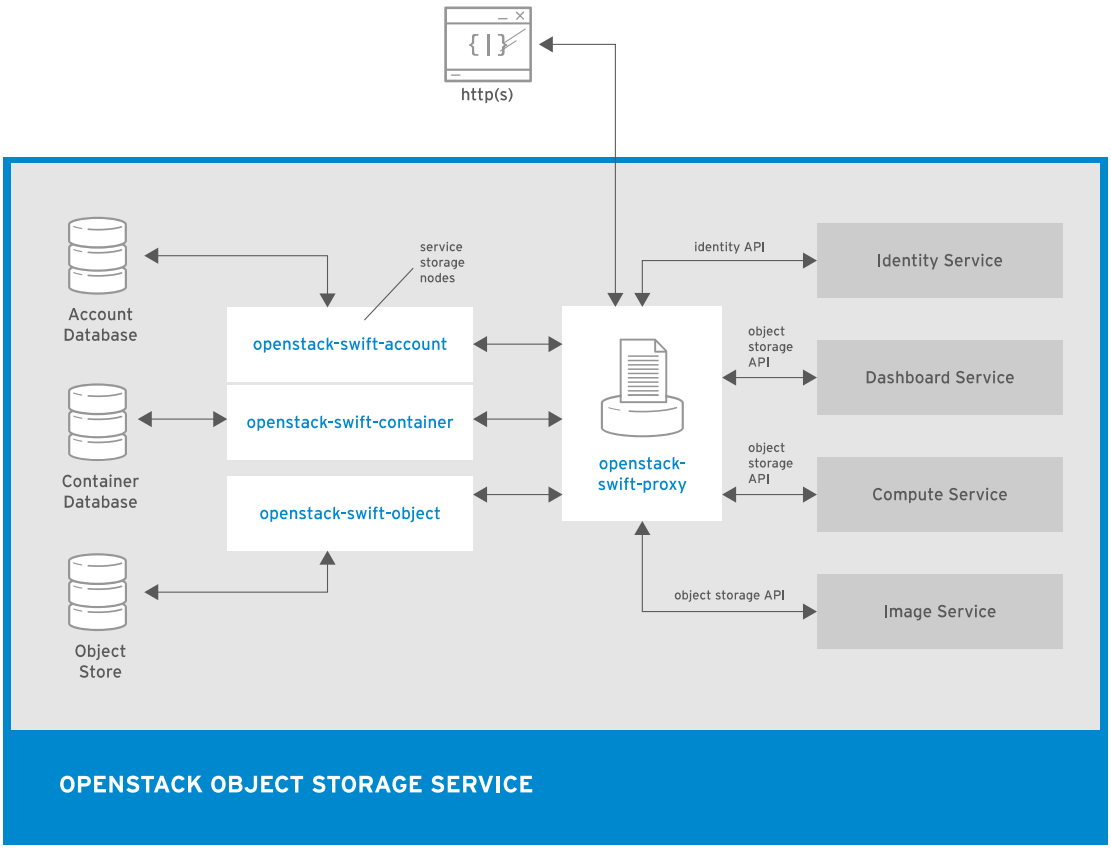


Figure 5.3: The OpenStack object store architecture

When a user invokes the object store service API, the proxy server demands an authentication token for the user to assert its right to access the object store. The identity service generates the token and returns it along with the object store service endpoint (URL) for the user to access. The user uses the token to identify themselves to the object store service. After successful authentication, the proxy server communicates with each of the Account, Container, and Object servers to determine the appropriate physical location for listing/creating an object-container (container) or object. The account, container, and object servers use their corresponding rings to

supply the required information to the proxy server. The proxy server performs the final action and returns the status of the operation to the end user.

More details regarding the rings are covered later in this section.

COMPARING CEPH WITH SWIFT FOR OBJECT STORAGE

Both Swift and Ceph are open source object storage systems. They both use standard hardware, allow scale-out storage, and are easy to deploy in enterprises of all sizes.

This is perhaps where the similarities end. Ceph lends itself to block access storage, transactional storage, and is recommended for single sites. Swift uses Object API access to storage, and is recommended for unstructured data and geographical distribution. Applications that mostly use block access storage are built in a different way from those that use object access storage. The decision might come down to which applications need object storage and how they access it.

Swift protects written data first and can therefore take additional time to update the entire cluster. Ceph does not do this, which makes it a better candidate for databases and real-time data. Swift would be a better choice for large-scale, geographically dispersed, unstructured data. This means that you might need or want both Ceph and Swift. This decision will depend on the types of applications, the geographical structure of your data centers, the type of objects that need to be stored, consistency of the data replicated, transactional performance requirements, and the number of objects to be stored.

The OpenStack object store (Swift) and Red Hat Ceph Storage have many architectural differences which make each of them a right fit for different needs. The following table outlines some of those differences.

Comparison of Swift and Ceph Object Storage

DIFFERENCE	OPENSTACK OBJECT STORE	RED HAT CEPH STORAGE
Access method	REST API only	REST API (RADOSGW), CephFS, RBD wrapped around the Ceph native API (librados)
Consistency model	Eventual	Full
Available storage formats	Object store only	File store, block store, and object store
Speed and latency	Traffic through proxy servers causes more overhead leading to higher latency.	Direct traffic to storage nodes allows better transfer speed and lower latency.

It is important to understand that none of the storage solutions, be it OpenStack object store or Red Hat Ceph Store, replace each other. Each of them has their applicable use case. Red Hat Ceph Storage fits better in single-site environments that deal more with transactional data, whereas the OpenStack object store works well with much larger environments dealing with large amount of data.

EVENTUAL CONSISTENCY AS AN ADVANTAGE

Eventual consistency is the ability to maintain the same versions of an object across the cluster within a reasonably short amount of time. Compare this to *strong* or *full consistency*, which is the ability to maintain the same versions of an object across the cluster at the same time. There is no storage solution that serves every need of the infrastructure. Unstructured data, such as photos, videos, text files, document files, and so on, which is experiencing heavy growth on the

internet, demands a storage system that can ensure the data is highly durable, available, and manageable in a cost-effective manner. Cost-effectiveness is based on factors such as cost/GB and the amount of effort to manage the data. According to Eric Brewer's CAP (Consistency, Availability, and Partition tolerance) theorem, you can simultaneously achieve up to two of the three among data consistency, availability, and partition tolerance. The OpenStack object store traded timely consistency for the combination of availability and partition tolerance.

THE RINGS

It is critical to understand the hierarchical organization of the data in the OpenStack object store system prior to understanding the rings. The OpenStack object store system accommodates data within the hierarchical arrangement of account, container, and object. The following describes the account, container, and object hierarchical levels.

- *Account* is where the entire hierarchy of containers and objects begins. It can be compared to the project or tenant in the Red Hat OpenStack Platform environment. It groups a set of containers that a specific user has access to on the basis of the account the user is identified with. Multiple containers may exist with the same name in different accounts.

- *Container* stores and isolates objects. The access control list is enforced at the container level and not at the object level. The storage policies are also applied on the containers. End users can create or delete containers. Containers with no access control list do not enforce any security restriction.

- *Object* stores the actual data content with its metadata. End users can upload objects as large as 5 GB. Objects with same name in separate containers may represent disparate data.

The following example describes the account, container, and object hierarchy. The OpenStack object store API represents the resource path in a special format as follows.

```
/v1/{account}/{container}/{object}
```

If you upload an object called **`object1`** to the `container1` container within the `654321` account, the resulting HTTPS URL of the object would be `https://myobjectserver.example.com:8080/v1/654321/container1/object1`. You need to use the same URL to interact with the object store API. Be aware that if your object name contains a slash character (/), such as **`category1/object1`**, then the object store will not create another level of hierarchy with the name `category1`. Instead, it will enable you to create a pseudo-folder for further isolation of objects. In this case, the resulting HTTPS URL would be `https://myobjectserver.example.com:8080/v1/654321/container1/category1/object1`.

The hierarchical path of an object gets an MD5 hashing performed on it and the generated hash is stored in a hash table. The hash tables are contained within *partitions*. The following discussion covers the details regarding partitions.

The OpenStack object store system uses the ring databases to compute the physical location of where a particular piece of data should be stored in the cluster. The rings maintain the records using the relationships between the elements, such as regions, zones, devices, partitions, and replicas. A *region* is a geographical isolation of the OpenStack object storage implementation. Each region has a separate server endpoint. A *zone* may represent a group of devices, servers, or even an entire data center. It is important to declare zones using a group of potentially independent failure domains so that the data in the object store system can survive in the event of failure. A *device* is a storage unit that has the back-end file system on it and is typically mounted under the **`/srv/node`** directory. A *partition* is the smallest unit that the object store services operate on to ensure data integrity and durability. All of the account, container, and object data gets placed into the partitions. These partitions are then distributed and replicated across the cluster.

While building the account, container, and object rings using **`swift-ring-builder`**, you need to specify the partition power, replica, and minimum time interval for partition reassignment (in hours). 2 to the partition power (2^partition power) gives the number of partitions (partition count). The partition power is derived from the number of bits kept from the MD5 hash of the object's path. Red Hat recommends an optimistic value for the partition count to ensure efficient data distribution and better scalability. For example, if the partition count is 64 in an environment of two devices within a single region and zone in the cluster, each of the devices will get 32 partitions. If the environment gets two additional devices then each device will get 16 partitions as the number of the partitions remains unchanged. The replica determines the number of copies of the partitions in the cluster. The default replica count of the OpenStack object store system is three, which means there will be three copies of an object in the cluster. Red Hat therefore recommends a minimum of three devices in the cluster for efficient distribution. The **`min_part_hours`** (minimum number of hours to wait for partition reassignment) causes the cluster to lock down the other replicas of an object while a partition moves around. This occurs in circumstances such as the addition of a new device to the cluster or replacing a faulty device in the cluster. The **`min_part_hours`** option allows the objects to remain available in the latest possible version while partitions are being moved. This is how the OpenStack object store system offers data durability, fault tolerance, and eventual consistency.

Among the fields of the ring data structure are arrays of:

- Devices that signify the device id, device name, region, zone, weight of the device, IP address of the storage server, the port that the storage server listens on, and additional metadata.
- Device id lists that signify the partition-to-device assignments.

BENEFITS, USE CASES, AND RECOMMENDED PRACTICES

Comparing Object Storage with Block Storage

One of the main differences between block storage and object storage is that a volume can only be accessed via instances, and by one instance at a time, whereas any instance or service can access the objects stored in containers because all objects stored within Swift have an accessible URL. Swift also supports the Amazon Simple Storage Service (S3) API.

The Benefits of Using Swift

Object storage has several distinct advantages over volume storage. As previously mentioned, it is accessible from any OpenStack service, it supports the Amazon S3 API, and is fully distributed.

The reduced cost can also be an advantage With object storage you only pay for the amount of storage that you use; that is, if you upload 5 GB then you only pay for 5 GB. With volume storage, you pay for the size of the disk you create. If you create a 50 GB volume, you will pay for all 50 GB whether or not it is all used. However, be aware that if you use Swift over multiple data centers then the cost can spiral because you are moving a lot of data over the internet.

Swift is best used for large pools of small objects. It is easily scalable, whereas volumes are not.

Use Cases

A major university uses Swift to store videos of all men's and women's sporting events. All events for an entire year are stored in an omnipresent and easily accessible storage solution. Students, alumni, and fans can use any internet-enabled web browser to access the university's web site and click a link to view their desired sporting event.

Recommended Practice: Disk Failure

It is Friday night and a drive has failed. Swift starts an automatic, self-healing, workaround by writing replicas to a hand-off node. Monday comes around and you change the failed drive, format

it, and mount it. The drive is, of course, empty. Swift automatically starts replicating data that is supposed to be in that zone. In this case, you do not even have to do anything to the ring because the physical drive was replaced; zones do not change and so there is no need to rebalance the ring.

> **NOTE**
> If you were to change the size of the physical drive, then you would have to rebalance the ring.

REMOVING AND REBALANCING ZONES

It is important to understand the concepts behind a storage system, to comprehend the policies, and to design and plan carefully before production.

Zones are created to ensure that failure is not an option. Each data replica should reside within a different zone. Zone configuration ensures that if one zone fails then there are still two up and running that can either accept new objects or retrieve stored objects.

The recommended number of zones is five, on five separate nodes. As mentioned previously, Swift, by default, writes three replicas. If there are only three zones and one becomes unavailable, Swift cannot hand off the replica to another node. With five nodes, Swift has options and can automatically write the replica to another node, ensuring that eventually there will be three replicas.

After Swift is set up and configured, it is possible to rectify or alter the storage policy. Extra devices can be added at any time.

Storage rings can be built on any hardware that has the appropriate version of Swift installed. Upon building or rebalancing (changing) the ring structure, the rings must be redistributed to include all of the servers in the cluster. The **`swift-ring-builder`** tool is used to build and manage rings.

To build the three rings for account, object, and container, the following syntax is used to add a new device to a ring:

```
[root@demo]# swift-ring-builder account.builder add
 zzone-ipaddress:6202/device weight
[root@demo]# swift-ring-builder container.builder add
 zzone-ipaddress:6201/device weight
[root@demo]# swift-ring-builder object.builder add
 zzone-ipaddress:6200/device weight
```

The *zone* includes a number as the ID for the rack to which the server belongs. The *ipaddress* is the IP address of the server. The *device* is the device partition to add. The *weight* includes the size of the device's partition.

> **NOTE**
> Prior to the Newton release of OpenStack, the Object service used ports 6002, 6001, and 6000 for the account, container, and object services. These earlier default Swift ports overlapped with ports already registered with IANA for X-Server, causing SELinux policy conflicts and security risks. Red Hat OpenStack Platform switched to the new ports in the Juno release, and the upstream Swift project completed the switch in Newton.

SWIFT COMMANDS

There are two sets of commands for Swift, an earlier version and a later version. The earlier commands, such as **`swift post`**, **`swift list`**, and **`swift stat`**, are still supported. However, OpenStack is moving to the OpenStack Unified CLI described below.

> **NOTE**
> By default, the following commands require the OpenStack user to have either the `admin` or `swiftoperator` roles.

The **`openstack container`** command is used to manage objects in OpenStack. The **`openstack container create`** command is used to create containers:

```
[user@demo ~]$ openstack container create cont1
```

The **`openstack container list`** command displays all containers available to the user:

```
[user@demo ~]$ openstack container list
+------------+
| Name       |
+------------+
| cont1      |
+------------+
```

The **`openstack container delete`** command deletes the specified container:

```
[user@demo ~]$ openstack container delete cont1
```

The **`openstack object create`** command uploads an existing object to the specified container:

```
[user@demo ~]$ openstack object create cont1 object.cont1
+--------------+-----------+----------------------------------+
| object       | container | etag                             |
+--------------+-----------+----------------------------------+
| object.cont1 | cont1     | d41d8cd98f00b204e9800998ecf8427e |
+--------------+-----------+----------------------------------+
```

The **`openstack container save`** command saves the contents of an existing container locally:

```
[user@demo ~]$ openstack container save cont1
[user@demo ~]$ ls -al
-rw-rw-r--. 1 user user   0 May 29 09:45 object.cont1
```

The **`openstack object list`** command lists all of the objects stored in the specified container:

```
[user@demo ~]$ openstack object list cont1
```

```
+--------------+
| Name         |
+--------------+
| object.cont1 |
+--------------+
```

The **`openstack object delete`** command deletes an object from the specified container:

```
[user@demo ~]$ openstack object delete cont1 object.cont1
```

CONFIGURATION AND LOG FILES

The following lists the object store configuration and log files.

/var/log/containers/swift and /var/log/containers/httpd/swift-*.
: Default location of all log entries.

/var/log/messages
: Location of all messages related to HAProxy and Swift configuration, and Swift CLI tool requests.

/var/lib/config-data/swift/etc/swift/object-server.conf
: Holds the configuration for the different back-end Swift services supporting replication (`object-replicator`), object information management in containers (`object-updater`), and object integrity (`object-auditor`).

OPENSTACK SERVICES THAT USE THE OBJECT STORE

Some of the services that use the OpenStack object store as the back-end storage provider are described below.

- The OpenStack image service uses the OpenStack object store as the storage back end to accommodate the instance images as objects. Storing images in the object store instead of the local file store offers high-availability and reliability of images. The OpenStack image service can also be configured to use Red Hat Ceph Storage as the back-end storage provider.

- The OpenStack block store service has several components, such as the API server, volume server, volume scheduler, and volume backup server. The volume backup server component maintains the volume backups which can be used to construct an identical volume. The OpenStack block store service's backup component uses the OpenStack object store as the storage back end to accommodate the archives of the persistent volumes that act as volume backups. This component can also be configured to use Red Hat Ceph Storage as the back-end storage provider.

- The OpenStack workflow service acts as a workflow management system in the Red Hat OpenStack Platform environment. A collection of tasks executed in a specific order is called a workflow. These workflows can be stored as objects when the OpenStack workflow service uses the OpenStack object store as the back-end storage provider. These workflows can involve tasks as simple as deploying or configuring an instance or as complex as a complete stack deployment plan.

- The OpenStack bare-metal provisioning service of the undercloud performs introspection on the overcloud nodes before deploying the overcloud. This introspection collects data about hardware characteristics of the overcloud nodes so that a specific category of node can be assigned a certain role. The introspected data gets stored in the OpenStack object store system of the undercloud.

- Red Hat OpenStack Platform has a newer multitenant messaging and notification service called Zaqar. Even though the functionality and features of Zaqar are analogous to other messaging systems such as RabbitMQ, ActiveMQ, and Qpid, it does not replace any of them. Instead, it is an easy-to-use choice for application developers that they can use to make the components of their applications communicate. Traditionally, MongoDB and Redis were the supported back-end data stores for Zaqar. Now, Zaqar has evolved far enough to use the OpenStack object store also as the back-end data store for storing its messages.

- The OpenStack metric service is based on a time-series database that stores metrics and resources published by the OpenStack data collection service. A time-series database is optimized for handling data that contains arrays of numbers indexed by time stamp. The OpenStack metric service provides a REST API to create or edit metric data. The `gnocchi-metricd` component of the metric service computes statistics, in real time, of received data. This computed data is stored and indexed for faster retrieval. The metric supports various back ends for storing the metric data and indexed data. The OpenStack object store is one of the supported storage back ends to store metric data. The default storage driver is `file`. An overcloud deployment uses the `ceph` storage driver as the storage for the metric data. The metric service uses a PostgreSQL or a MySQL database to store indexed data and any associated metadata. The default storage driver for indexed data is `PostgreSQL`.

REFERENCES

Further information is available in the *Red Hat Ceph Storage for the Overcloud Guide* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en/red-hat-openstack-platform/

COMPARING OBJECT STORES

In this exercise, you will add a second disk to the OpenStack object store. You will also create an object container with a file object in it and find the structure of the object in the back-end disk.

OUTCOMES

You should be able to:

- Create objects in the OpenStack object store and find the actual bits on the disk.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab storage-swift setup`** to verify that OpenStack services are running and the environment is ready for the exercise.

```
[student@workstation ~]$ lab storage-swift setup
```

1. Verify the health of the Docker containers relevant to OpenStack object store.

   1.1. Open an SSH session to `controller0` as `root`.

```
[student@workstation ~]$ ssh root@controller0
[root@controller0 ~]#
```

   1.2. Verify the health of the containers relevant to OpenStack object store.

```
[root@controller0 ~]# docker ps --format="{{.Names}}\t{{.Status}}" | grep swift
swift_proxy Up About an hour (healthy)
swift_container_auditor Up About an hour
swift_object_expirer  Up About an hour
swift_object_updater  Up About an hour
swift_container_replicator  Up About an hour
swift_account_auditor Up About an hour
swift_account_server  Up About an hour (healthy)
swift_object_replicator Up About an hour
swift_container_server  Up About an hour (healthy)
swift_rsync Restarting (11) 49 minutes ago
swift_account_reaper  Up About an hour
swift_account_replicator  Up About an hour
swift_object_auditor  Up About an hour
swift_object_server Up About an hour (healthy)
swift_container_updater Up About an hour
```

NOTE
You can ignore the **Restarting** state of the `swift_rsync` container in the preceding output.

2. Add **/srv/node/d2** as an additional device to OpenStack object store. Update the ring files to ensure that each device in the OpenStack object store gets to accommodate a copy of the same object.

NOTE
Due to the classroom limitations, you are using a directory (**/srv/node/d2**) in place a disk device to support the OpenStack object store. In practice, you would use an actual disk device, typically formatted with a file system such as XFS.

2.1. Verify that there is currently a single device that the OpenStack object store ring files are aware of.

```
[root@controller0 ~]# cd /var/lib/config-data/puppet-generated/swift/etc/swift/
[root@controller0 swift]# swift-ring-builder account.ring.gz
Note: using account.builder instead of account.ring.gz as builder file
account.builder, build version 2, id 5ef67cb074a0430ebc5d25f1ce51f329
1024 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 devices, 0.00 balance,
 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 1 (0:00:00
 remaining)
The overload factor is 0.00% (0.000000)
Ring file account.ring.gz is up-to-date
Devices:   id region zone ip address:port replication ip:port  name weight
 partitions balance flags meta
            0      1    1 172.24.4.1:6002     172.24.4.1:6002    d1 100.00
 1024    0.00
[root@controller0 swift]# swift-ring-builder container.ring.gz
Note: using container.builder instead of container.ring.gz as builder file
container.builder, build version 2, id 66dcd204b5f44375b5d4e1dd677a962b
1024 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 devices, 0.00 balance,
 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 1 (0:00:00
 remaining)
The overload factor is 0.00% (0.000000)
Ring file container.ring.gz is up-to-date
Devices:   id region zone ip address:port replication ip:port  name weight
 partitions balance flags meta
            0      1    1 172.24.4.1:6001     172.24.4.1:6001    d1 100.00
 1024    0.00
[root@controller0 swift]# swift-ring-builder object.ring.gz
Note: using object.builder instead of object.ring.gz as builder file
object.builder, build version 2, id 8a94f604449f4d83aeb2c18b0bd47f79
1024 partitions, 1.000000 replicas, 1 regions, 1 zones, 1 devices, 0.00 balance,
 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 1 (0:00:00
 remaining)
The overload factor is 0.00% (0.000000)
```

```
Ring file object.ring.gz is up-to-date
Devices:   id region zone ip address:port replication ip:port  name weight
 partitions balance flags meta
            0      1    1 172.24.4.1:6000     172.24.4.1:6000    d1 100.00
 1024    0.00
```

2.2. Update the OpenStack object store ring builder files to add **/srv/node/d2** as an additional device. Set the weight of the new device to 100.

```
[root@controller0 swift]# swift-ring-builder account.builder \
add z1-172.24.4.1:6002/d2 100
WARNING: No region specified for z1-172.24.4.1:6002/d2. Defaulting to region 1.
Device d1r1z1-172.24.4.1:6002R172.24.4.1:6002/d2_"" with 100.0 weight got id 1
[root@controller0 swift]# swift-ring-builder container.builder \
add z1-172.24.4.1:6001/d2 100
WARNING: No region specified for z1-172.24.4.1:6001/d2. Defaulting to region 1.
Device d1r1z1-172.24.4.1:6001R172.24.4.1:6001/d2_"" with 100.0 weight got id 1
[root@controller0 swift]# swift-ring-builder object.builder \
add z1-172.24.4.1:6000/d2 100
WARNING: No region specified for z1-172.24.4.1:6000/d2. Defaulting to region 1.
Device d1r1z1-172.24.4.1:6000R172.24.4.1:6000/d2_"" with 100.0 weight got id 1
```

2.3. Set the replica count to 2 in the ring builder files.

```
[root@controller0 swift]# swift-ring-builder account.builder set_replicas 2
The replica count is now 2.000000.
The change will take effect after the next rebalance.
[root@controller0 swift]# swift-ring-builder container.builder set_replicas 2
The replica count is now 2.000000.
The change will take effect after the next rebalance.
[root@controller0 swift]# swift-ring-builder object.builder set_replicas 2
The replica count is now 2.000000.
The change will take effect after the next rebalance.
```

2.4. Verify the new device and the replica count in the ring builder files.

```
[root@controller0 swift]# swift-ring-builder account.builder
account.builder, build version 3, id 5ef67cb074a0430ebc5d25f1ce51f329
1024 partitions, 2.000000 replicas, 1 regions, 1 zones, 2 devices, 100.00 balance,
 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 1 (0:00:00
 remaining)
The overload factor is 0.00% (0.000000)
Ring file account.ring.gz is obsolete
Devices:   id region zone ip address:port replication ip:port  name weight
 partitions balance flags meta
            0      1    1 172.24.4.1:6002     172.24.4.1:6002    d1 100.00
 1024    0.00
            1      1    1 172.24.4.1:6002     172.24.4.1:6002    d2 100.00
  0 -100.00
[root@controller0 swift]# swift-ring-builder container.builder
container.builder, build version 3, id 66dcd204b5f44375b5d4e1dd677a962b
1024 partitions, 2.000000 replicas, 1 regions, 1 zones, 2 devices, 100.00 balance,
 0.00 dispersion
```

```
The minimum number of hours before a partition can be reassigned is 1 (0:00:00
 remaining)
The overload factor is 0.00% (0.000000)
Ring file container.ring.gz is obsolete
Devices:   id region zone ip address:port replication ip:port  name weight
 partitions balance flags meta
            0      1    1 172.24.4.1:6001     172.24.4.1:6001    d1 100.00
 1024    0.00
            1      1    1 172.24.4.1:6001     172.24.4.1:6001    d2 100.00
  0 -100.00
[root@controller0 swift]# swift-ring-builder object.builder
object.builder, build version 3, id 8a94f604449f4d83aeb2c18b0bd47f79
1024 partitions, 2.000000 replicas, 1 regions, 1 zones, 2 devices, 100.00 balance,
 0.00 dispersion
The minimum number of hours before a partition can be reassigned is 1 (0:00:00
 remaining)
The overload factor is 0.00% (0.000000)
Ring file object.ring.gz is obsolete
Devices:   id region zone ip address:port replication ip:port  name weight
 partitions balance flags meta
            0      1    1 172.24.4.1:6000     172.24.4.1:6000    d1 100.00
 1024    0.00
            1      1    1 172.24.4.1:6000     172.24.4.1:6000    d2 100.00
  0 -100.00
```

2.5. Rebalance the OpenStack object store ring files.

```
[root@controller0 swift]# swift-ring-builder account.builder rebalance
Reassigned 1024 (100.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
[root@controller0 swift]# swift-ring-builder container.builder rebalance
Reassigned 1024 (100.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
[root@controller0 swift]# swift-ring-builder object.builder rebalance
Reassigned 1024 (100.00%) partitions. Balance is now 0.00.  Dispersion is now 0.00
```

2.6. Restart the `swift_proxy`, `swift_account_server`, `swift_container_server`, and `swift_object_server` containers.

```
[root@controller0 swift]# docker restart \
swift_proxy swift_account_server \
swift_container_server swift_object_server
swift_proxy
swift_account_server
swift_container_server
swift_object_server
```

2.7. Log off from `controller0`.

```
[root@controller0 swift]# logout
Connection to controller0 closed.
[student@workstation ~]$
```

3. Create the object container `finance-container1` in the `finance` project. Upload **`/home/student/finance-object1`** to `finance-container1` as the `finance-`

object1 object. Locate the file representing the finance-object1 object in the back-end storage devices of the OpenStack object store.

3.1. As developer1, create the object container finance-container1.

```
[student@workstation ~]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack container create \
finance-container1
+------------------+--------------------+----------------+
| account          | container          | x-trans-id     |
+------------------+--------------------+----------------+
| AUTH_3c00...953c | finance-container1 | txe0c6b...30a9 |
+------------------+--------------------+----------------+
```

3.2. Upload **/home/student/finance-object1** to finance-container1.

```
[student@workstation ~(developer1-finance)]$ openstack object create \
finance-container1 finance-object1
+-----------------+--------------------+----------------------------------+
| object          | container          | etag                             |
+-----------------+--------------------+----------------------------------+
| finance-object1 | finance-container1 | 740580d95587be0197cf16e2d96f4931 |
+-----------------+--------------------+----------------------------------+
```

3.3. Locate finance-object1 in the back-end storage devices of the OpenStack object store.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
find /srv/node -iname *.data
/srv/node/d1/objects/211/7d6/34cd...b7d6/1541157229.63432.data
/srv/node/d2/objects/211/7d6/34cd...b7d6/1541157229.63432.data
```

Notice the two copies of the data file under each of the **/srv/node/d1** and **/srv/node/d2** directories that pose as storage devices to the OpenStack object store in the classroom environment. These data files back the finance-object1 object in the finance-container1 object container.

3.4. Confirm that the content of the data files as shown in the preceding output matches that of the finance-object1 object.

```
[student@workstation ~(developer1-finance)]$ cat finance-object1
This is to test the OpenStack Object Store service
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
'find /srv/node -iname *.data -exec cat {} \;'
This is to test the OpenStack Object Store service
This is to test the OpenStack Object Store service
```

4. Upload another object, **/home/student/finance-object2**, to finance-container1 as finance-object2. Trace the back-end data files to verify the copies of the object. Determine the appropriate replicas for each object.

4.1. Upload **/home/student/finance-object2** to finance-container1.

```
[student@workstation ~(developer1-finance)]$ openstack object create \
finance-container1 finance-object2
```

```
...output omitted...
```

4.2. Trace the back-end data files and confirm that each of the `/srv/node/d1` and `/srv/node/d2` storage devices contain a copy of the objects.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
find /srv/node -iname *.data
/srv/node/d1/objects/211/7d6/34cd...b7d6/1541494418.07269.data
/srv/node/d1/objects/150/f97/25ac...0f97/1541494485.92954.data
/srv/node/d2/objects/211/7d6/34cd...b7d6/1541494418.07269.data
/srv/node/d2/objects/150/f97/25ac...f97/1541494485.92954.data
```

Notice the similarity in the hierarchy of each pair of the data files, spanning across both the storage devices. This similarity helps determine the data files that belong to the same object. For example, both **`/srv/node/d1/`** and **`/srv/node/d2/`** have a copy of the **`1541494418.07269.data`** data file under the same **`objects/211/7d6/34cd...b7d6/`** hierarchy stating that the data files are the replicas of the same object.

Cleanup

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

```
[student@workstation ~]$ lab storage-swift cleanup
```

This concludes the guided exercise.

MANAGING SHARED FILE SYSTEMS

OBJECTIVES

After completing this section, students should be able to describe the multiple methods for configuring the new shared file systems component.

THE SHARED FILE SYSTEMS SERVICE (MANILA)

The Shared File Systems service enables tenant users to provision and manage network-accessible shared file systems. Tenant users access the shares from instances as remote file system mounts. The service provides a solution for automating provisioning and management of legacy file-based storage resources for scalable cloud applications, while still maintaining efficient multitenancy isolation. The service operates on the control plane rather than the data plane, meaning that the service facilitates share management functions after the share is provisioned. The service is not involved in the data transfer between clients and the back-end storage after mounting. This data plane decoupling enables clients to directly access the back-end storage after the share is provisioned, using the metadata information of the share.

The OpenStack Shared File Systems service is comprised of the following processes:

manila-api
: The API server exposes the Shared File Systems service functions to the tenant users.

manila-scheduler
: The scheduler is selects which storage back end will serve the share creation being requested.

manila-share
: This service coordinates with the back-end storage systems that accommodate the shares. It has two modes of operation: with or without handling of share servers. With the handling of share servers mode enabled, it can manage the life cycle of the back end share servers.

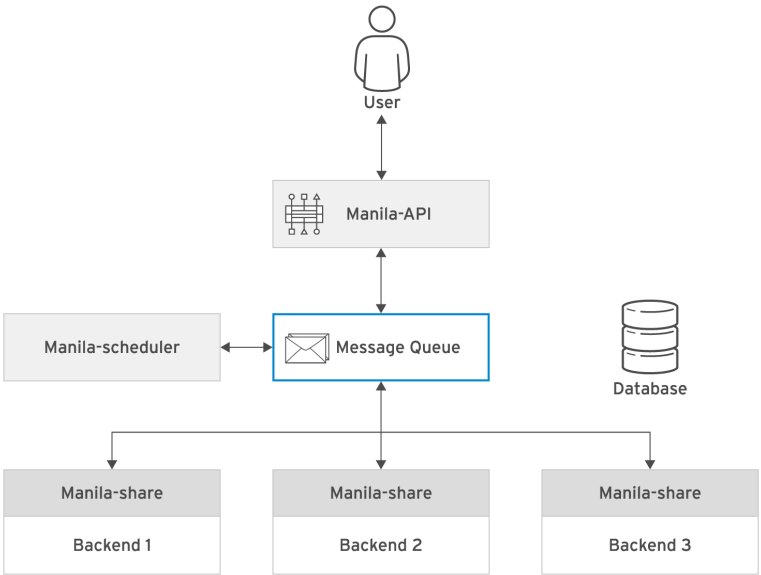


Figure 5.4: The Shared File Systems service architecture

SHARED FILE SYSTEMS SERVICE RESOURCES

The Shared File Systems service manages several resources, including:

- Share: A *share* is an instance of a persistent file-based storage unit. Users of compute instances can access the share as a remote file system. These shares can be configured as either read-only or read-write.

- Share Type: A *share type* is used to classify the shares on the basis of the characteristics the share's back-end storage provider. The share type allows multiple back-end storage providers to be active at the same time. For example, administrators can create separate share types for different storage back ends, such as CephFS, GlusterFS. The share type also allows the Shared File Systems service to use separate installations of the same storage solution. For example, there can be separate instances of Red Hat Ceph Storage clusters with each cluster serving different data centers across the cloud. In such cases, administrators can create the share types with the back ends set to use different instances of the Ceph clusters for different types of share.

- Share Network: The *share network* binds a tenant network with the tenant subnet to which the shares are attached. When you create a share, you can specify the share network for the share. Remember to create the share network in the same project as the share. Share networks allow tenant users to provision shares that are accessible only from a desired set of instances that participate in the same network segment. Network security restrictions can be enforced on the shares depending on their connection to the share networks.

- Share Access Rules: The *share access rules* define the access rights a user has on a particular share. For example, access rules can be defined such that a specific cephx user has read and write access on a certain CephFS share.

Through the use of drivers, the Shared File Systems service can be configured to use multiple storage solutions, including some of the proprietary ones, as the back-end storage providers for the shares. There is also a generic driver for the Shared File Systems service that uses persistent volumes from the Block Storage service to accommodate the shares. However, this generic approach of accommodating the shares is not a recommended practice in production. It is critical to understand that each driver has a separate set of configuration parameters and requires the `manila-share` service to be configured accordingly. The Shared File Systems service supports several storage back ends, including the following:

- CephFS
- GlusterFS
- NetApp Clustered Data ONTAP
- Hitachi NAS Platform File Services

NFS-GANESHA MEDIATING STORAGE ACCESS

NFS-Ganesha is an implementation of the user-space NFS service that acts as the mediator between the tenant users' machines and the back-end storage of the shared file system. The system on which the `nfs-ganesha` service runs becomes the gateway server. The gateway server can be configured to allow or deny the tenant users to access a specific shared file system. The tenant users use NFS clients to request access to the gateway server and the gateway server in turn connects them to the shared file system if policies permit.

The implementation of NFS-Ganesha uses plug-ins to create a File System Abstraction Layer (FSAL). This plug-in-oriented architecture of NFS-Ganesha allows it to easily extend to several file-system back ends, including, but not limited, to CephFS, GlusterFS, and ZFS. NFS-Ganesha also supports various NFS protocol versions, such as NFSv2, v3, v4, v4.1, v4.2, and pNFS, which provides room for extension in future use cases.

When a client requests to mount a share, the NFS-Ganesha server verifies the access rights of the requester set on the share and allows or denies the request accordingly. If it allows a client to

mount the share, any operation the client invokes on the share is redirected to the appropriate back-end storage provider via the related FSAL. NFS-Ganesha is also compatible with the Kerberos authentication system to implement Kerberos-enabled NFS shares.

Shared File Systems on Gluster via NFS-Ganesha

There are two approaches to using Red Hat Gluster Storage as the back-end storage provider for the OpenStack Shared File Systems service. The first approach, using the GlusterFS native driver, supports the GlusterFS share protocol and enforces access control via TLS certificates. Clients need to import the appropriate certificates and should support the GlusterFS share protocol to access the shared file system. The second approach, using th GlusterFS (NFS/CIFS) driver, uses NFS-Ganesha to expose the Gluster-backed shared file systems to NFS/CIFS clients. The clients that support the NFS or CIFS share protocol can easily mount the Gluster-backed shared file systems without having to worry about supporting the native GlusterFS share protocol. The second approach implements IP-based access control policies for the tenants to access the shared file systems.

Shared File Systems on Ceph via NFS-Ganesha

The Ceph File System (CephFS) is a file-based interface to a Ceph cluster. It provides a scalable, POSIX-compliant, parallel file system which stores its data and metadata as objects in Red Hat Ceph Storage. CephFS depends on a cluster node running a Metadata Server (MDS) to coordinate access to a Ceph cluster and to manage metadata related to its files.

The CephFS NFS driver implements NFS-Ganesha for Ceph-backed shared file systems. Tenant users can create shares in CephFS and access them using NFS 4.1 via NFS-Ganesha. The NFS-Ganesha server enforces the access controls on the shares which dictate which clients can access the shares. As the NFS-Ganesha node acts as the gateway server, any NFS-compatible client can access the share as a friendly NFS share. The clients do not need to support the native CephFS share protocol.

As Red Hat Ceph Storage acts as the unified storage solution for block-based, object-based, and file-based storage requirements, administrators do not need to worry about setting up a separate storage cluster to serve the block-based and object-based storage requirements. Rather, they can use the same storage cluster to serve all of the aforementioned types of storage requirements. This optimizes the efficiency of the storage cluster. Moreover, the use of separate networks for data plane network traffic and control plane network traffic decouples the tenant user's network from the storage network, which enhances the security of the storage network.

Shared File Systems on Ceph via CephFS Native Driver

The CephFS native driver for the OpenStack Shared File Systems service allows the shared file systems to be exported via the CephFS share protocol. Clients interested in mounting such shares should have the native Ceph client installed. The CephFS native driver uses the cephx authentication system to identify and authorize users that can access the shared file systems. Based on the preconfigured access policies of the shared file systems for the cephx users, the CephFS native driver coordinates with the Red Hat Ceph Storage cluster services to update the capabilities of the cephx user to grant the requested operation. Clients must use the appropriate user ID and secret key while mounting the shared file system or the requested mount operation will fail.

MANAGING SHARED FILE SYSTEMS USING THE CEPHFS NATIVE DRIVER

The following steps describe how to configure the OpenStack Shared File Systems service to use the CephFS native driver.

Firstly, verify the status of the `manila-api`, `manila-scheduler`, and the `manila-share` processes. Use the **`docker`** commands to verify the status of the containerized Red Hat OpenStack Platform services as opposed to the **`systemctl`** commands for the systemd unit-based services.

Verify that the configuration settings in the **`manila.conf`** file is configured appropriately. Among the INI sections of the **`manila.conf`** file that require modifications are **`[DEFAULT]`** and **`[backend_name]`**.

The parameters **`enabled_share_backends`** and **`enabled_share_protocols`** under the **`[DEFAULT]`** INI section should be set to **`cephfs`** and **`CEPHFS`** respectively.

You should also define an additional INI section with its name matching the back-end name. This INI section typically exists at the end of the file. This INI section has the CephFS native driver-specific parameters set to appropriate values. Among the parameters is the **`share_backend_name`** parameter, which should match the name of the back end as defined under the **`[DEFAULT]`** INI section. Set the **`share_driver`** parameter to **`manila.share.drivers.cephfs.driver.CephFSDriver`**. Set the correct path to the Ceph configuration file using the **`cephfs_conf_path`** parameter. Use the **`cephfs_auth_id`** parameter to set the correct ID as in the appropriate cephx user account representing the service user account for the OpenStack Shared File Systems service. Set the cluster name to **`ceph`** using the **`cephfs_cluster_name`** parameter. Set the **`cephfs_protocol_helper_type`** to **`CEPHFS`** for enabling CephFS as the export share protocol.

As the administrative user, create the share type using the **`manila type-create`** command.

```
[user@demo ~]$ manila type-create cephfstype false
```

The **`false`** argument in the preceding command represents the value of the **`driver_handles_share_servers`** parameter. If you wish to dynamically scale the share nodes, set this value to **`true`**.

As a tenant user, create the share using the **`manila create`** command.

```
[user@demo ~]$ manila create --name demo-share \
--share-type cephfstype cephfs 1 \
```

The **`--share-type`** option helps to classify the shares based on the back-end storage providers. In this example, the value 1 indicates that the size of the share would be 1 GB.

When the share is available, launch the instance from where you intend to access the share with the separate network interfaces for each of the tenant network and the storage network.

Create the cephx user using the **`ceph auth get-or-create`** command on the Ceph Monitor node. Ensure you save the secret key of the user. This user will be used to authenticate while mounting the CephFS share in the instance.

```
[root@controller ~]# ceph --name=client.manila \
--keyring=/etc/ceph/ceph.client.manila.keyring \
auth get-or-create client.cloud-user > /root/cloud-user.keyring
```

The preceding command creates the `client.cloud-user` cephx user while authenticating as the `client.manila` user, using the key-ring file **`/etc/ceph/ceph.client.manila.keyring`** that has the secret key for the user.

Copy the secret key and the Ceph configuration file to the instance from where you intend to mount the share.

Grant the appropriate access rights for the cephx user on the share using the **`manila access-allow`** command.

```
[user@demo ~]$ manila access-allow demo-share cephx cloud-user
```

Trace the export path location of the share using the **`manila share-export-location-list`** command.

```
[user@demo ~]$ manila share-export-location-list \
demo-share --columns Path
```

Use the native Ceph client to mount the share with its appropriate export path location.

REFERENCES

Further information is available in the OpenStack Shared File System service with CephFS via NFS chapter of the *CephFS via NFS Back End Guide for the Shared File System Service* for Red Hat OpenStack Platform at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/

MANAGING SHARED FILE SYSTEMS

In this exercise, you will create a shared file system and access it from an instance.

OUTCOMES

You should be able to create and use a shared file system with Red Hat Ceph Storage as its storage provider.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab storage-manila setup`** to ensure that the environment is ready for the exercise. This command creates additional items that you are going to use at different steps of the exercise.

```
[student@workstation ~]$ lab storage-manila setup
```

1. From `workstation`, open an SSH session to `controller0` as `root`.

```
[student@workstation ~]$ ssh root@controller0
[root@controller0 ~]#
```

2. Verify the state and the configuration settings of the OpenStack Shared File Systems service.

   2.1. On `controller0`, verify that the containers relevant to the Shared File Systems service are active.

```
[root@controller0 ~]# docker ps --format="{{.Names}}\t{{.Status}}" | grep manila
openstack-manila-share-docker-0 Up 25 minutes
manila_api  Up 28 minutes
manila_scheduler  Up 28 minutes (healthy)
```

   2.2. On `controller0`, verify the configuration settings of the Shared File Systems service that intends to use the Red Hat Ceph Storage in the classroom environment as the storage provider.

```
[root@controller0 ~]# crudini --get \
/var/lib/config-data/puppet-generated/manila/etc/manila/manila.conf \
DEFAULT enabled_share_backends
cephfs
[root@controller0 ~]# crudini --get \
/var/lib/config-data/puppet-generated/manila/etc/manila/manila.conf \
DEFAULT enabled_share_protocols
CEPHFS
[root@controller0 ~]# tail \
```

```
/var/lib/config-data/puppet-generated/manila/etc/manila/manila.conf
[cephfs]
driver_handles_share_servers=False
share_backend_name=cephfs
share_driver=manila.share.drivers.cephfs.driver.CephFSDriver
cephfs_conf_path=/etc/ceph/ceph.conf
cephfs_auth_id=manila
cephfs_cluster_name=ceph
cephfs_enable_snapshots=False
cephfs_protocol_helper_type=CEPHFS
```

NOTE

In general, the configuration file of the Shared File Systems service is **/etc/manila/manila.conf**. However, in the containerized overcloud, service configuration files are located under the **/var/lib/config-data/puppet-generated/<service>/etc/<service>/** directory. Make the configuration changes in files under this directory to adjust settings for containerized services. To implement the changes, restart or reload the containers.

2.3. Log off from `controller0`.

```
[root@controller0 ~]# logout
Connection to controller0 closed.
[student@workstation ~]$
```

2.4. As the overcloud user `architect1`, verify the status of the Shared File Systems service instances. Viewing the status of the Shared File Systems service instances requires administrative privileges.

```
[student@workstation ~]$ source ~/architect1-finance-rc
[student@workstation ~(architect1-finance)]$ manila service-list \
--columns Binary,Status,State
+------------------+---------+-------+
| Binary           | Status  | State |
+------------------+---------+-------+
| manila-scheduler | enabled | up    |
| manila-share     | enabled | up    |
+------------------+---------+-------+
```

3. Create the share `finance-share1` of 1 GB capacity in the `finance` project. Create the `cephfstype` share type and use it for `finance-share1`.

3.1. As `architect1`, create the share type `cephfstype`. Set the value of **driver_handles_share_servers** to **false**.

```
[student@workstation ~(architect1-finance)]$ manila type-create cephfstype false
+----------------------+--------------------------------------+
| Property             | Value                                |
+----------------------+--------------------------------------+
| required_extra_specs | driver_handles_share_servers : False |
| Name                 | cephfstype                           |
| Visibility           | public                               |
| is_default           | -                                    |
| ID                   | bd3ae330-2170-4e57-a54f-865faacde5f1 |
```

```
| optional_extra_specs |                                      |
| Description          | None                                 |
+----------------------+--------------------------------------+
```

IMPORTANT
To dynamically scale the share nodes, set **`driver_handles_share_servers`** to **`true`**.

Managing share types requires administrative rights. Load the `architect1` user's credentials using the **`source /home/student/architect1-finance-rc`** command.

3.2. As `developer1`, create the new share `finance-share1`.

```
[student@workstation ~(architect1-finance)]$ source developer1-finance-rc
[student@workstation ~(developer1-finance)]$ manila create \
--name finance-share1 --share-type cephfstype cephfs 1
...output omitted...
```

3.3. Verify that the new share is available.

```
[student@workstation ~(developer1-finance)]$ manila list \
--columns Name,'Share Proto',Status,'Share Type Name'
+----------------+-------------+-----------+-----------------+
| Name           | Share Proto | Status    | Share Type Name |
+----------------+-------------+-----------+-----------------+
| finance-share1 | CEPHFS      | available | cephfstype      |
+----------------+-------------+-----------+-----------------+
```

4. Launch the `finance-server1` instance using two network interfaces. The first interface is connected to the tenant network `finance-network1`. The second interface is connected to the storage provider network `provider-storage`, allowing the instance to directly reach the shared file system data network.

4.1. As `developer1`, launch the instance with the following parameters.

- **`Name`**: finance-server1
- **`Flavor`**: default
- **`Image`**: rhel7
- **`SSH Keypair`**: example-keypair
- **`Network for first NIC`**: finance-network1
- **`Network for second NIC`**: provider-storage
- **`User data file`**: /home/student/manila/user-data.file

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--nic net-id=finance-network1 \
--nic net-id=provider-storage \
--user-data /home/student/manila/user-data.file \
finance-server1 --wait
```

```
...output omitted...
```

4.2. Add the available floating IP address to the instance.

```
[student@workstation ~(developer1-finance)]$ openstack floating \
ip list -c 'Floating IP Address'
+---------------------+
| Floating IP Address |
+---------------------+
| 172.25.250.107      |
+---------------------+
[student@workstation ~(developer1-finance)]$ openstack server \
add floating ip finance-server1 172.25.250.107
```

4.3. Verify that the instance is reachable from `workstation`.

```
[student@workstation ~(developer1-finance)]$ ping -c 3 172.25.250.107
PING 172.25.250.107 (172.25.250.107) 56(84) bytes of data.
64 bytes from 172.25.250.107: icmp_seq=1 ttl=63 time=1.76 ms
64 bytes from 172.25.250.107: icmp_seq=2 ttl=63 time=1.04 ms
64 bytes from 172.25.250.107: icmp_seq=3 ttl=63 time=0.587 ms

--- 172.25.250.107 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2005ms
rtt min/avg/max/mdev = 0.587/1.132/1.763/0.483 ms
```

5. Create the exclusive cephx user `client.cloud-user` to access the Ceph-backed share `finance-share1`. Allow the `client.cloud-user` cephx user to read from and write to the share. The OpenStack Shared File Systems service uses the `client.manila` cephx user to authenticate as while communicating with the Ceph cluster, although it has more privileges than what an end user actually requires to access the share. You are going to use the **cephx** authentication mechanism for enabling access to the Ceph-backed share `finance-share1`.

5.1. Open an SSH session to `controller0` as `root`.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0
[root@controller0 ~]#
```

5.2. Create the **cephx** user `client.cloud-user`. Save the key ring to **`/root/cloud-user.keyring`**.

```
[root@controller0 ~]# ceph --name=client.manila \
--keyring=/etc/ceph/ceph.client.manila.keyring \
auth get-or-create client.cloud-user > /root/cloud-user.keyring
```

5.3. Log off from `controller0`.

```
[root@controller0 ~]# logout
Connection to controller0 closed.
```

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

5.4. Copy the **ceph.conf** and **cloud-user.keyring** files to the `finance-server1` instance launched in one of the preceding steps.

```
[student@workstation ~(developer1-finance)]$ cd manila
[student@workstation manila(developer1-finance)]$ scp \
root@controller0:{cloud-user.keyring,/etc/ceph/ceph.conf} .
cloud-user.keyring                              100%   68    36.3KB/s   00:00
ceph.conf                                       100%  797   805.3KB/s   00:00
[student@workstation manila(developer1-finance)]$ scp \
{cloud-user.keyring,ceph.conf} cloud-user@172.25.250.107:
```

5.5. Add and verify the access rights to `finance-share1` for `client.cloud-user`.

```
[student@workstation ~(developer1-finance)]$ manila access-allow \
finance-share1 cephx cloud-user
+--------------+--------------------------------------+
| Property     | Value                                |
+--------------+--------------------------------------+
| access_key   | None                                 |
| share_id     | c28b9123-ae9a-4951-a6d1-5b856c83a600 |
| created_at   | 2018-10-11T10:15:22.000000           |
| updated_at   | None                                 |
| access_type  | cephx                                |
| access_to    | cloud-user                           |
| access_level | rw                                   |
| state        | queued_to_apply                      |
| id           | c4665a6b-7570-4221-b3cb-623d0bbef87d |
+--------------+--------------------------------------+
[student@workstation ~(developer1-finance)]$ manila access-list \
finance-share1 --columns access_to,access_level,state
+------------+--------------+--------+
| Access_To  | Access_Level | State  |
+------------+--------------+--------+
| cloud-user | rw           | active |
+------------+--------------+--------+
```

If you omit the **--access-level** option, **rw** is used as the default access level.

6. Access the `finance-share1` share from the `finance-server1` instance.

6.1. Open an SSH session to `finance-server1` as `cloud-user`.

```
[student@workstation ~(developer1-finance)]$ ssh cloud-user@172.25.250.107
[cloud-user@finance-server1 ~]$
```

6.2. Create the mount point **/mnt/ceph** for the share.

```
[cloud-user@finance-server1 ~]$ su -
Password: redhat
```

```
[root@finance-server1 ~]# mkdir /mnt/ceph
```

6.3. Open another terminal as `student` on `workstation`. As the `developer1` tenant user, determine the export location of `finance-share1`.

```
[student@workstation ~]$ source developer1-finance-rc
[student@workstation ~(developer1-finance)]$ manila share-export-location-list \
finance-share1 --columns Path
+-----------------------------------------------------------------------+
| Path                                                                  |
+-----------------------------------------------------------------------+
| 172.24.3.1:6789:/volumes/_nogroup/cea022a9-c00c-4003-b6f3-8fea2a49bd5f |
+-----------------------------------------------------------------------+
```

Make a note of the export path. You need to use the export path while mounting the share.

6.4. Enable the Ceph repositories on `finance-server1` and install the *ceph-fuse* package.

```
[root@finance-server1 ~]# curl -s -f \
-o /etc/yum.repos.d/ceph.repo http://materials.example.com/ceph.repo
[root@finance-server1 ~]# yum install ceph-fuse
...output omitted...
Is this ok [y/d/N]: y
...output omitted...
Complete!
```

6.5. Mount the share on **`/mnt/ceph`** from `finance-server1`. Use the export path noted from the earlier step. Verify that the share is successfully mounted.

```
[root@finance-server1 ~]# ceph-fuse /mnt/ceph/ \
--id=cloud-user --conf=/home/cloud-user/ceph.conf \
--keyring=/home/cloud-user/cloud-user.keyring \
--client-mountpoint=/volumes/_nogroup/cea022a9-c00c-4003-b6f3-8fea2a49bd5f
2018-10-26 00:20:28.786368 7fa70ae140c0 -1 init, newargv = 0x55e83fad2d80
 newargc=9
ceph-fuse[1487]: starting ceph client
ceph-fuse[1487]: starting fuse
[root@finance-server1 ~]# df -Th
Filesystem     Type            Size  Used Avail Use% Mounted on
...output omitted...
ceph-fuse      fuse.ceph-fuse  1.0G     0  1.0G   0% /mnt/ceph
```

6.6. Create a text file in the **`/mnt/ceph`** directory.

```
[root@finance-server1 ~]# cd /mnt/ceph
[root@finance-server1 ceph]# echo hello > hello.txt
[root@finance-server1 ceph]# cat hello.txt
```

```
hello
```

This preceding commands confirm that the `finance-share1` share is readable and writable based on the access rights set in one of the earlier steps.

6.7. Log off from `finance-server1`.

```
[root@finance-server1 ceph]# logout
[cloud-user@finance-server1 ~]$ logout
[student@workstation ~(developer1-finance)]$
```

7. Terminate the `finance-server1` instance and delete the `finance-share1` share.

7.1. Terminate `finance-server1`.

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

7.2. Delete `finance-share1`.

```
[student@workstation ~(developer1-finance)]$ manila delete finance-share1
```

7.3. Delete the share type `cephfstype`.

```
[student@workstation ~(developer1-finance)]$ source architect1-finance-rc
[student@workstation ~(architect1-finance)]$ manila type-delete cephfstype
```

Cleanup

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

```
[student@workstation ~]$ lab storage-manila cleanup
```

This concludes the guided exercise.

MANAGING EPHEMERAL AND PERSISTENT STORAGE

OBJECTIVES

After completing this section, students should be able to interpret the behavior of ephemeral storage configuration choices.

EPHEMERAL VERSUS PERSISTENT STORAGE

The following table outlines the primary differences between ephemeral and the persistent storage units.

Comparison of Ephemeral and Persistent Storage Units

DIFFERENCE	EPHEMERAL STORAGE	PERSISTENT STORAGE
Existence	Is created when the instance is created.	Is created upon successful execution of a user's API request.
Persistence	Does not persist after the instance is terminated.	Persists after the instance is terminated until the tenant user manually deletes it.
Speed and Latency	Offers lower latency with improved responsiveness, when configured using direct attached storage.	Offers higher latency but with capacity to handle larger I/O workloads.
Maximum size	Depends on the available storage capacity.	Depends on the applicable project quota for the storage capacity.
Sizing	Performed using the attributes of the instance flavor.	Performed based on the user's request.

When a tenant user creates an instance using an image, the only initial storage that the instance has is the ephemeral storage. It has its own advantages and disadvantages. It is a very simple and easy-to-use storage resource that lacks advanced features such as storage tiering and disk capacity expansion. Any data stored in ephemeral storage does not survive after its instance is terminated. The ephemeral device lifecycle is managed by the OpenStack compute service.

Persistent storage, on the other hand, is a volume that continues to exist even after instance to which it is attached is terminated. Data stored in a persistent volume survives even after the deletion of the instance on which the data was created. It supports the storage tiering and capacity expansion features. Persistent storage, based on the type of storage requirements it serves, is managed by the Block Storage, Object Store, and Shared File Systems services.

EPHEMERAL STORAGE AND LIBVIRT

The compute service uses the native libvirt virtualization platform of the Red Hat Enterprise Linux host on which it is installed with KVM as the hypervisor to manage the life cycle of the instances in the back end. The instances exist as KVM virtual machines and any tenant user's API instruction gets translated to the `libvirtd` service via the Libvirt compute driver. The `libvirtd` service then acts on the virtual machine as per the user's instruction. For example, if the user submits the request to launch an instance via the OpenStack Compute service API, the native `libvirtd`

service of the compute node takes the required action to define a KVM virtual machine and acknowledge the user's request via the Compute service API.

The flavor determines the size of each instance. The default settings of the OpenStack compute service let the instances consume compute and storage resources from the hypervisor system. However, in practice the instances use a shared storage pool to back their ephemeral storage requirements. Among the flavor attributes are the number of vCPUs, amount of memory, amount of swap memory, size of the instance's disk, and an additional ephemeral disk size of the instance. The storage units that get created as part of the flavor definition are all ephemeral in nature.

By default, the compute service creates the back-end objects/files representing the ephemeral storage under the **`/var/lib/nova/instances/`*`UUID`*`/`** directory on the compute node. If Red Hat Ceph Storage is being used as the back-end storage provider for the OpenStack Compute service, the back-end objects are stored within a Ceph pool typically named `vms`. It is critical to understand that using Red Ceph Storage as the back-end storage provider for the OpenStack Compute service does not convert the ephemeral storage to persistent storage. The ephemeral storage will continue to get revoked when the instance is deleted irrespective of whether it is accommodated locally in the compute node or if it uses a centralized storage pool from an external back-end storage provider such as Red Hat Ceph Storage.

CHOOSING AN INSTANCE STORAGE OPTION

There are three main strategies used while providing instances with the ephemeral storage based on factors such as IOPS requirements, performance requirements, and administration requirements affecting the running workloads. The strategies involve:

- Running compute nodes with shared file systems on separate servers.

  This approach leads to running instances with their disks on nodes separate from the compute nodes. In this case, if the compute nodes go offline with intentional or unintentional actions and they do not have any running instances, the rest of the cloud will not be impacted. Because the storage functionality is decoupled from the compute nodes, the storage servers will continue to operate as usual and offer the storage resources to the rest of the cloud.

  Among the advantages of this approach are easy recovery for instances during compute node failure, flexibility to scale the storage capacity, and operational simplicity with the dedicated storage system. While there are advantages, there are disadvantages as well using this strategy, such as performance degradation due to network access to storage, and impact on I/O performance due to traffic from other unrelated instances.

- Running compute nodes with shared filesystem on compute nodes.

  This option requires that each compute node have significant storage capacity. A distributed file system is configured across the compute nodes to allow each of their storage capacity to be contributed to a single unit and used as a single mount point. This option allows OpenStack to scale using a third-party storage solution if circumstances demand.

  Among the disadvantages of this option is that the maximum number of disks that can be added is limited to what the chassis size of the compute node allows. Also, the availability of the storage resources lessens with the functional number of compute nodes.

- Running compute nodes with no shared file systems.

  In this approach, the compute nodes store the instances' disks locally. This approach allows instances to directly perform I/O operations on the disks without getting the network involved, resulting in improved performance.

  But, in this approach, the data on the instances of the compute nodes becomes irrecoverable in case of a node failure. There is no option for scalability to serve the additional storage

requirements in this approach. Even if you vertically scale the capacity of the compute nodes, you will be limited to the maximum capacity that the chassis size of the node allows.

The strategy of running a shared file system on a storage server decoupled from the compute node fits well in circumstances where reliability and scalability are the priorities. On the other hand, if you are concerned more about storage performance and less about data persistence, you may stick to the second or third of the aforementioned strategies.

As with the ephemeral storage of the instances, the persistent storage of the instances also has its own applicability and use cases. Persistent storage is useful in dealing with data that gets reused. Among the implementations of persistent storage in the Red Hat OpenStack Platform environment are volumes from the Block Storage service, object containers from the Object Store service and shares from the Shared File Systems service. Each of the volumes, object containers, and shares serve specific storage requirements. The volumes are mainly for the instances to consume. The volumes do not have the wide level of access as the object containers do but have more scope in terms of the native security features that the object containers do not have. The object containers are meant to be fault tolerant and are more popular in service-oriented infrastructure design. The share is also a persistent storage that the instances can access as remote file systems using network share protocols such as NFS, CIFS, CephFS, and GlusterFS.

CREATING PERSISTENT ROOT DISKS

The contents of an instance image can be extracted to a persistent volume just like you can extract to the ephemeral storage of the instance. However, any change made to the root file system via any instance will survive until the tenant user deletes the persistent volume. This is also an approach of constructing identical instances using the persistent root file systems.

The following command allows you to create a persistent volume with the contents of the root file system in it.

```
[user@demo ~]$ openstack volume create --size 10 --image rhel7 demo-vol
```

The preceding command creates a volume `demo-vol` with its size set to 10 GB. The `rhel7` image is used as the source of the root file system contents to extract into the volume. This volume can be referenced with the **`--volume`** option of the **`openstack server create`** command to launch an instance using the persistent root file system volume.

UNDERSTANDING INSTANCE MIGRATION

Migrating an instance means moving a virtual machine from one compute node to the other compute node. Migration can help ensure that the services running in an instance remain available even in case of a compute node failure. Migration also helps to perform scheduled maintenance of the compute nodes.

The two fundamental types of migration are cold and live. In a cold migration, the instances are shut down and then migrated to another compute node. The default configuration settings along with the SSH public keys exchanged between the compute nodes are sufficient to initiate a non-live migration of instances. The exchange of the SSH public keys between the compute nodes ensure that the `libvirtd` service in each node can communicate with its peer. This communication results in creation of a tunnel through which the instance migration occurs in a secure (encrypted) manner.

In the live migration, the instance continues to run while being migrated to another compute node. Live migration fits well for the workloads that require the least possible to zero downtime. Such workloads require the applications in the instances to run without potential disruption. The live migration is further classified into the following categories:

- Shared storage-based live migration. In this type of migration, the in-memory data of the instance gets copied to the destination compute node. The ephemeral storage of the instance is accommodated in the same back-end storage shared between the source compute node and the destination compute node and so the data in the instance's ephemeral storage does not need to be transferred to the destination compute node.

- Block live migration. In this type of migration, the in-memory data of the instance along with the data in the instance's ephemeral storage moves from one compute node to another. The reason for the requirement of this transfer is that the source and the destination compute nodes do not share the same back-end storage to accommodate the ephemeral storage of the instances. Instead, the compute nodes store a local copy of the instance's ephemeral storage. This type of migration takes longer to complete because of the increased network load due to data from both the memory pages of the instance as well as the ephemeral storage.

- Volume-backed live migration. This type of migration involve instances that use the persistent volumes rather than using the ephemeral disks. In this type of migration, only the in-memory data of the instance gets transferred to the destination compute node as the volumes stay as usual in the storage nodes and do not need to be moved along with the instance. This saves the network from being more heaviliy loaded and improves the latency.

REFERENCES

Further information is available in the *Differences between storage types* section of the *OpenStack Architecture Design Guide* available at
https://docs.openstack.org/arch-design/design-storage/design-storage-concepts.html

Further information is available in the *Storage Management* section of the Libvirt documentation available at
https://libvirt.org/storage.html

Further information is available in the *Instance storage solutions* section of the *OpenStack Architecture Design Guide* available at
https://docs.openstack.org/arch-design/design-compute/design-compute-storage.html

MANAGING EPHEMERAL AND PERSISTENT STORAGE

In this exercise, you will launch instances using ephemeral and persistent root disks, attach additional disks to the instances, and trace the objects representing the disks in the back-end storage.

OUTCOMES

You should be able to:

- Create instances using ephemeral and persistent root disks and add additional disks.
- Compare and contrast storage structures and locations for those different disks.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab storage-compare setup`** to ensure that the environment is ready for the exercise. This command creates additional items that you are going to use at different stages of the exercise.

```
[student@workstation ~]$ lab storage-compare setup
```

1. On `workstation`, launch the `finance-server1` instance in the `finance` project and trace the back-end objects representing the storage of the instance.

   1.1. As `architect1`, launch the instance with the following parameters.

   - **`name`**: **`finance-server1`**
   - **`flavor`**: **`default`**
   - **`image`**: **`rhel7`**
   - **`key-name`**: **`example-keypair`**
   - **`nic`**: **`net-id=finance-network1`**
   - **`availability-zone`**: **`nova:compute1.overcloud.example.com`**

```
[student@workstation ~]$ source architect1-finance-rc
[student@workstation ~(architect1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--nic net-id=finance-network1 \
--availability-zone nova:compute1.overcloud.example.com \
finance-server1 --wait
...output omitted...
```

   1.2. As `developer1`, find the unique ID of the `finance-server1` instance. Note the unique ID of the instance. You are going to use this ID in the following step.

```
[student@workstation ~(architect1-finance)]$ source developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack server list \
-c ID -c Name
+--------------------------------------+-----------------+
| ID                                   | Name            |
+--------------------------------------+-----------------+
| 2c477594-d47a-4f44-9974-17a1ee0f266d | finance-server1 |
+--------------------------------------+-----------------+
```

1.3. Match the unique ID of the instance noted from the preceding step with the back-end object in the Ceph pool `vms`.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
rados -p vms \
ls | grep 2c477594-d47a-4f44-9974-17a1ee0f266d
rbd_id.2c477594-d47a-4f44-9974-17a1ee0f266d_disk
```

2. Delete the `finance-server1` instance and verify that the root disk of the instance no longer exists in the `vms` Ceph pool.

2.1. Delete the `finance-server1` instance.

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

2.2. Confirm that no object exists in the `vms` Ceph pool to represent the root disk of the `finance-server1` instance. Use the same ID of the instance as in the preceding step.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
rados -p vms \
ls | grep 2c477594-d47a-4f44-9974-17a1ee0f266d
```

3. Explore the configuration settings of the OpenStack Compute service that allows it to use the Ceph storage to accommodate the instance disks. Using Ceph storage as the back end for the OpenStack compute service saves you from flooding the local disks of the compute node and running into scalability issues. Instead, it stores the disks into the `vms` Ceph pool and offers better scalability as with the Ceph storage.

3.1. Open an SSH session to `compute1` as `root`.

```
[student@workstation ~(developer1-finance)]$ ssh root@compute1
[root@compute1 ~]#
```

3.2. View the settings that enable Ceph storage as the back-end storage provider for the OpenStack Compute service.

```
[root@compute1 ~]# grep rbd \
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf
rbd_user=openstack
rbd_secret_uuid=945a02d4-c931-11e8-b4cd-52540001fac8
images_type=rbd
images_rbd_pool=vms
```

```
images_rbd_ceph_conf=/etc/ceph/ceph.conf
```

4. Adjust the configuration settings of the OpenStack Compute service to use the default approach of storing the instance disks on the local compute node rather than the Ceph storage. Launch an instance and verify that the root disk of the instance exists locally on the compute node. Delete the instance and verify that the root disk of the instance no longer exists on the compute node.

   4.1. Comment out the **`images_type=rbd`** line under the **`[libvirt]`** INI section of the **`/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf`** file.

```
[libvirt]
...output omitted...
#images_type=rbd
...output omitted...
```

   4.2. Restart the `nova_compute` container to bring the configuration change into effect.

```
[root@compute1 ~]# docker restart nova_compute
nova_compute
```

   4.3. Wait for thirty seconds and verify the status of the `nova_compute` container.

```
[root@compute1 ~]# sleep 30; docker ps \
--format="{{.Names}}\t{{.Status}}" | grep nova
nova_compute  Up 31 seconds (healthy)
nova_migration_target Up 4 hours
nova_libvirt  Up 4 hours
nova_virtlogd Up 4 hours
```

   4.4. Log off from `compute1`.

```
[root@compute1 ~]# logout
[student@workstation ~(developer1-finance)]$
```

   4.5. As `architect1`, launch the instance with the following parameters.

   - **`name`**: **`finance-server1`**
   - **`flavor`**: **`default`**
   - **`image`**: **`rhel7`**
   - **`key-name`**: **`example-keypair`**
   - **`nic`**: **`net-id=finance-network1`**
   - **`availability-zone`**: **`nova:compute1.overcloud.example.com`**

```
[student@workstation ~(developer1-finance)]$ source architect1-finance-rc
[student@workstation ~(architect1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--nic net-id=finance-network1 \
--availability-zone nova:compute1.overcloud.example.com \
finance-server1 --wait
```

```
...output omitted...
```

4.6. As `developer1`, find the unique ID of the `finance-server1` instance. Note the unique ID of the instance. You are going to use this ID in the following step.

```
[student@workstation ~(architect1-finance)]$ source developer1-finance-rc
[student@workstation ~(developer1-finance)]$ openstack server list \
-c ID -c Name
+--------------------------------------+-----------------+
| ID                                   | Name            |
+--------------------------------------+-----------------+
| 54ec33d2-961c-4cc1-91b5-5d336fcb336a | finance-server1 |
+--------------------------------------+-----------------+
```

4.7. Match the unique ID of the instance noted from the preceding step with the name of the directory under the **`/var/lib/nova/instances`** path on `compute1`. Confirm that the directory contains the root disk image file of the `finance-server1` instance.

```
[student@workstation ~(developer1-finance)]$ ssh root@compute1 \
ls /var/lib/nova/instances
54ec33d2-961c-4cc1-91b5-5d336fcb336a
_base
compute_nodes
locks
[student@workstation ~(developer1-finance)]$ ssh root@compute1 \
ls /var/lib/nova/instances/54ec33d2-961c-4cc1-91b5-5d336fcb336a
console.log
disk
disk.info
```

The file **`disk`** represents the root disk of the `finance-server1` instance.

5. Delete the `finance-server1` instance and verify that the root disk of the instance no longer exists on the compute node.

5.1. Delete the `finance-server1` instance.

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

5.2. Confirm that the root disk image file of the `finance-server1` instance no longer exists in the compute node.

```
[student@workstation ~(developer1-finance)]$ ssh root@compute1 \
ls /var/lib/nova/instances
_base
compute_nodes
locks
```

6. You have dealt with the ephemeral root disk of an instance in the preceding steps. The following steps of the exercise guide you on how to use a persistent root disk in an instance. On `controller0`, view the settings of the OpenStack Block Storage service that assists in using Ceph storage to accommodate persistent volumes. Create a persistent volume using

the `rhel7` image. Confirm that the volume exists in the `volumes` Ceph pool. Launch an instance with the new volume as its root disk.

6.1. View the back-end storage provider of the OpenStack Block Storage service.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
crudini --get \
/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf \
DEFAULT enabled_backends
tripleo_ceph
```

6.2. Open an SSH session to `controller0`.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0
```

6.3. View the settings under the **`tripleo_ceph`** INI section of the OpenStack Block Storage service configuration file.

```
[root@controller0 ~]# grep -A 6 \
'^\[tripleo_ceph\]$' \
/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf
[tripleo_ceph]
backend_host=hostgroup
volume_backend_name=tripleo_ceph
volume_driver=cinder.volume.drivers.rbd.RBDDriver
rbd_ceph_conf=/etc/ceph/ceph.conf
rbd_user=openstack
rbd_pool=volumes
```

Notice the Ceph pool that accommodates the volumes is set to **`volumes`**.

6.4. Log off from `controller0`.

```
[root@controller0 ~]# logout
[student@workstation ~(developer1-finance)]$
```

6.5. As `developer1`, create a volume using the `rhel7` image. Verify that the volume is created successfully.

```
[student@workstation ~(developer1-finance)]$ openstack volume create \
--size 10 \
--image rhel7 \
finance-vol1
...output omitted...
[student@workstation ~(developer1-finance)]$ openstack volume list
+-------------+-------------+-----------+------+-------------+
| ID          | Name        | Status    | Size | Attached to |
+-------------+-------------+-----------+------+-------------+
| 1972...34cf | finance-vol1 | available |   10 |             |
```

```
+-------------+--------------+-----------+------+-------------+
```

> **NOTE**
> After executing the **`openstack volume create`** command, wait for up to two minutes while the contents from the `rhel7` image are extracted to the `finance-vol1` volume. You may also use the **`watch openstack volume list`** command to monitor the real-time status of the `finance-vol1` volume.

Copy the unique ID of the volume from the output of the preceding **`openstack volume list`** command to compare with the object in the `volumes` Ceph pool. The unique ID assists in identifying the volumes that the `volumes` Ceph pool accommodates.

6.6. Verify that the Ceph pool `volumes` has an object representing the volume `finance-vol1`. Use the unique ID of the volume noted in the preceding step.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
rados -p volumes \
ls | grep 19723579-a453-4412-bf1c-377d647e34cf
rbd_id.volume-19723579-a453-4412-bf1c-377d647e34cf
```

6.7. As `developer1`, launch the instance with the following parameters.

- **`name`**: **`finance-server2`**
- **`flavor`**: **`default`**
- **`key-name`**: **`example-keypair`**
- **`nic`**: **`net-id=finance-network1`**
- **`volume`**: **`finance-vol1`**

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--volume finance-vol1 \
--key-name example-keypair \
--nic net-id=finance-network1 \
finance-server2 --wait
...output omitted...
```

7. Access the new instance `finance-server2` using any of the available floating IP addresses. Create a text file with arbitrary text in it. Delete the instance.

7.1. View the available floating IP addresses.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip list\
-c "Floating IP Address"
+---------------------+
| Floating IP Address |
+---------------------+
| 172.25.250.109      |
+---------------------+
```

7.2. Associate the floating IP address with the `finance-server2` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server add \
```

```
floating ip finance-server2 172.25.250.109
```

7.3. Open an SSH session to the `finance-server2` instance and create a file as `root` with arbitrary text in it.

```
[student@workstation ~(developer1-finance)]$ ssh 172.25.250.109
[cloud-user@finance-server2 ~]$ sudo -i
[root@finance-server2 ~]# echo Testing >> /test.txt
[root@finance-server2 ~]# sync
```

IMPORTANT

The **sync** command flushes contents from the Linux buffer cache to the disk. Skipping this command may cost you the contents of the text file you created.

7.4. Log off from the `finance-server2` instance and delete it. Verify that the volume still exists in the `volumes` Ceph pool. Use the same unique ID of the volume as in the preceding steps.

```
[root@finance-server2 ~]# logout
[cloud-user@finance-server2 ~]$ logout
[student@workstation ~(developer1-finance)]$ openstack server \
delete finance-server2
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
rados -p volumes \
ls | grep 19723579-a453-4412-bf1c-377d647e34cf
rbd_id.volume-19723579-a453-4412-bf1c-377d647e34cf
```

Notice that the volume exists irrespective of the termination of the instance. This acts as the persistent storage of the instance.

8. Create a fresh instance using the same `finance-vol1` as the boot disk of the instance and confirm that the file you created with the arbitrary text in one of the preceding steps still exists.

8.1. As `developer1`, launch the instance with the following parameters.

- **name**: **`finance-server2`**
- **flavor**: **`default`**
- **key-name**: **`example-keypair`**
- **nic**: **`net-id=finance-network1`**
- **volume**: **`finance-vol1`**

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--volume finance-vol1 \
--key-name example-keypair \
--nic net-id=finance-network1 \
finance-server2 --wait
...output omitted...
```

8.2. Associate the available floating IP address with the `finance-server1` instance.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip list \
-c "Floating IP Address"
```

```
+---------------------+
| Floating IP Address |
+---------------------+
| 172.25.250.109      |
+---------------------+
[student@workstation ~(developer1-finance)]$ openstack server add \
floating ip finance-server2 172.25.250.109
```

8.3. Open an SSH session to the `finance-server1` instance and verify the contents of the **`/test.txt`** file.

```
[student@workstation ~(developer1-finance)]$ ssh 172.25.250.109
[cloud-user@finance-server2 ~]$ cat /test.txt
Testing
```

8.4. Log off from `finance-server2`.

```
[cloud-user@finance-server2 ~]$ logout
[student@workstation ~(developer1-finance)]$
```

9. Create a new volume of 1 GB in capacity and attach it to the `finance-server2` instance.

9.1. As `developer1`, create a 1 GB empty volume called `finance-vol2`.

```
[student@workstation ~(developer1-finance)]$ openstack volume create \
--size 1 finance-vol2
...output omitted...
```

9.2. Attach the `finance-vol2` volume to the `finance-server2` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server add \
volume finance-server2 finance-vol2
```

9.3. Verify the new disk of the `finance-server2` instance.

```
[student@workstation ~(developer1-finance)]$ ssh 172.25.250.109 lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    253:0    0  10G  0 disk
└─vda1 253:1    0  10G  0 part /
vdb    253:16   0   1G  0 disk
```

10. Confirm that there is a new volume object in the `volumes` Ceph pool representing the `finance-vol2` volume.

10.1. Find and record the ID of the `finance-vol2` volume to compare with the `volumes` Ceph pool object.

```
[student@workstation ~(developer1-finance)]$ openstack volume list \
-c ID -c Name
+--------------------------------------+--------------+
| ID                                   | Name         |
+--------------------------------------+--------------+
| 93b29d6b-59dd-4903-a0a0-42773ee02b89 | finance-vol2 |
| 19723579-a453-4412-bf1c-377d647e34cf | finance-vol1 |
```

```
+-------------------------------------+--------------+
```

10.2. Verify that the ID of the `finance-vol2` matches one of the volume objects in the `volumes` Ceph pool.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
rados -p volumes \
ls | grep 93b29d6b-59dd-4903-a0a0-42773ee02b89
rbd_id.volume-93b29d6b-59dd-4903-a0a0-42773ee02b89
```

11. Delete the `finance-server2` instance. Delete the `finance-vol1` and `finance-vol2` volumes. Confirm that the volumes no longer exist in the `volumes` Ceph pool.

11.1. Delete the `finance-server2` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server delete \
finance-server2
```

11.2. Delete the `finance-vol1` and `finance-vol2` volumes.

```
[student@workstation ~(developer1-finance)]$ openstack volume delete \
finance-vol1 finance-vol2
```

11.3. Confirm that the volumes no longer exist in the `volumes` Ceph pool.

```
[student@workstation ~(developer1-finance)]$ ssh root@controller0 \
rados -p volumes ls
rbd_directory
rbd_info
```

Cleanup

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

```
[student@workstation ~]$ lab storage-compare cleanup
```

This concludes the guided exercise.

MANAGING STORAGE

PERFORMANCE CHECKLIST

In this lab, you will manage persistent volumes for OpenStack compute instances. You will also manage shared file systems and use them from OpenStack compute instances.

OUTCOMES

You should be able to:

- Administer OpenStack block storage.
- Administer OpenStack shared file systems.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab storage-review setup`**, which verifies OpenStack services and previously created resources. This script also ensures that the required resources to proceed with the exercise are in place.

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

1. As `operator1`, create a 10 GB bootable volume called `production-volume1` in the `production` project using the `rhel7` image.
2. As `operator1`, create an empty 1 GB volume called `production-volume2` in the `production` project.
3. Verify that the new volumes exist as objects in the `volumes` Ceph pool.

> **NOTE**
> The OpenStack installation in the classroom environment uses Red Hat Ceph Storage as the back-end storage provider for the OpenStack Block Storage service.

4. As `operator1`, launch an instance with the following parameters in the `production` project using the `production-volume1` bootable volume.

   - **`name`**: **`production-server1`**
   - **`flavor`**: **`default`**
   - **`key-name`**: **`operator1-keypair`**
   - **`nic`**: **`net-id=production-network1`**
   - **`volume`**: **`production-volume1`**

   Attach the empty `production-volume2` volume to the `production-server1` instance.
5. Access the `production-server1` instance and verify that the empty `production-volume2` volume was successfully attached to the `production-server1` instance.

6. Create a 1 GB partition in `production-volume2` that appears as the device `vdb` in the partition table of the `production-server1` instance. Format the partition with the `xfs` file system type and mount it temporarily on **`/mnt`**. Create the file **`/mnt/hello.txt`** with **`hello`** as the text message in it. The file **`/mnt/hello.txt`** gets accommodated into the `production-volume2` as **`/mnt`** acts as the mount point for the volume. Also, create the file **`/hello2.txt`** with **`hello2`** as the text message in it. The file **`/hello2.txt`** gets accommodated into the `production-volume1` because you are using the volume as the root file system in the `production-server1` instance.

7. Delete the existing `production-server1` instance and create a fresh instance with `production-volume1` as the bootable volume using the following parameters.

   - `name`: **`production-server1`**
   - `flavor`: **`default`**
   - `key-name`: **`operator1-keypair`**
   - `nic`: **`net-id=production-network1`**
   - `volume`: **`production-volume1`**

   Attach the `production-volume2` volume to the new `production-server1` instance.

8. Confirm that the text files you created earlier persist, with all of the content in the `production-server1` instance.

9. As `operator1`, create a 1 GB shared file system in the `production` project. Set the name of the shared file system to `production-share1`, and the type to `cephfstype`. Use the `cephfs` protocol while creating `production-share1`. For administrative operations such as managing the types of the shared file systems, use the `architect1` tenant user.

> **NOTE**
> The preinstalled Red Hat Ceph Storage in the classroom environment backs the `production-share1` shared file system. The OpenStack Shared File Systems service uses the appropriate Ceph driver to coordinate with Red Hat Ceph Storage.

10. Attach an additional NIC from the `provider-storage` network to the `production-server1` instance in order to reach the network segment of the `production-share1` shared file system from the `production-server1` instance. Use DHCP as the configuration method for the additional NIC.

11. Create the cephx user `client.cloud-user` and save the key ring as **`cloud-user.keyring`**. Authenticate as `client.manila` using the key ring from **`/etc/ceph/ceph.client.manila.keyring`** while creating `client.cloud-user`. Transfer the appropriate Ceph configuration file and the **`cloud-user.keyring`** file to `production-server1`. Install any required utilities on `production-server1` to access the Ceph-backed shared file system using the repository configuration file from `http://materials.example.com/ceph.repo`.

12. Grant **`read`** and **`write`** permissions on `production-share1` to `client.cloud-user`.

13. Access the `production-share1` shared file system from the `production-server1` instance. Authenticate as `client.cloud-user` using the key ring from **`/home/cloud-user/cloud-user.keyring`** while accessing `production-share1`. The appropriate Ceph cluster settings are in **`/home/cloud-user/ceph.conf`**. Create the file **`hello.txt`** with **`hello`** as the text message in the shared file system.

14. Delete the `production-server1` instance and create a new instance with the following parameters:

- **`name`**: **`production-server2`**
- **`flavor`**: **`default`**
- **`key-name`**: **`operator1-keypair`**
- **`nic`**: **`net-id=production-network1`**
- **`nic`**: **`net-id=provider-storage`**
- **`volume`**: **`production-volume1`**
- **`user-data`**: **`/home/student/manila/user-data.file`**

> **NOTE**
> The user data file **`/home/student/manila/user-data.file`** contributes to configuring the second NIC of the instance.

Access the `production-share1` shared file system from the new `production-server2` instance to confirm that the text file you created in the shared file system in a previous step persists.

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab storage-review cleanup
```

This concludes the lab.

MANAGING STORAGE

PERFORMANCE CHECKLIST

In this lab, you will manage persistent volumes for OpenStack compute instances. You will also manage shared file systems and use them from OpenStack compute instances.

OUTCOMES

You should be able to:

- Administer OpenStack block storage.
- Administer OpenStack shared file systems.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab storage-review setup`**, which verifies OpenStack services and previously created resources. This script also ensures that the required resources to proceed with the exercise are in place.

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

1. As `operator1`, create a 10 GB bootable volume called `production-volume1` in the `production` project using the `rhel7` image.

   1.1. Source the `operator1` user's credentials.

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

   1.2. Create a 10 GB bootable volume called `production-volume1` using the `rhel7` image.

```
[student@workstation ~(operator1-production)]$ openstack volume create \
--image rhel7 \
--size 10 \
production-volume1
...output omitted...
```

> **NOTE**
> After executing the **`openstack volume create`** command, wait for up to two minutes while the contents from the `rhel7` image is extracted to the `production-vol1` volume. You may also use the **`watch openstack volume list`** command to monitor the real-time status of the `production-vol1` volume.

   1.3. Verify that the `production-volume1` volume is available.

```
[student@workstation ~(operator1-production)]$ openstack volume list \
-c Name \
-c Status
+--------------------+-----------+
| Name               | Status    |
+--------------------+-----------+
| production-volume1 | available |
+--------------------+-----------+
```

2. As `operator1`, create an empty 1 GB volume called `production-volume2` in the `production` project.

   2.1. Create an empty 1 GB volume called `production-volume2`.

```
[student@workstation ~(operator1-production)]$ openstack volume create \
--size 1 \
production-volume2
...output omitted...
```

   2.2. Verify that the `production-volume2` volume is available.

```
[student@workstation ~(operator1-production)]$ openstack volume list \
-c Name \
-c Status
+--------------------+-----------+
| Name               | Status    |
+--------------------+-----------+
| production-volume2 | available |
| production-volume1 | available |
+--------------------+-----------+
```

3. Verify that the new volumes exist as objects in the `volumes` Ceph pool.

> **NOTE**
> The OpenStack installation in the classroom environment uses Red Hat Ceph Storage as the back-end storage provider for the OpenStack Block Storage service.

   3.1. Find the unique IDs of `production-volume1` and `production-volume2`. Use these IDs to compare with the objects in the `volumes` Ceph pool.

```
[student@workstation ~(operator1-production)]$ openstack volume list \
-c Name \
-c ID
+--------------------------------------+--------------------+
| ID                                   | Name               |
+--------------------------------------+--------------------+
| 0b4944e7-6939-4350-86a4-71746c6fe4f3 | production-volume2 |
| 03c4ebcd-1829-4f08-8bdd-9d8e8dcf14ff | production-volume1 |
```

```
+-------------------------------------+--------------------+
```

3.2. Verify that the `production-volume1` volume exists as an object in the `volumes` Ceph pool. Use the same ID of `production-volume1` as in the preceding output.

```
[student@workstation ~(operator1-production)]$ ssh root@controller0 \
rados -p volumes \
ls | grep 03c4ebcd-1829-4f08-8bdd-9d8e8dcf14ff
rbd_id.volume-03c4ebcd-1829-4f08-8bdd-9d8e8dcf14ff
```

3.3. Verify that the `production-volume2` volume exists as an object in the `volumes` Ceph pool. Use the same ID of `production-volume2` as in the preceding output.

```
[student@workstation ~(operator1-production)]$ ssh root@controller0 \
rados -p volumes \
ls | grep 0b4944e7-6939-4350-86a4-71746c6fe4f3
rbd_id.volume-0b4944e7-6939-4350-86a4-71746c6fe4f3
```

4. As `operator1`, launch an instance with the following parameters in the `production` project using the `production-volume1` bootable volume.

- **`name`**: **`production-server1`**
- **`flavor`**: **`default`**
- **`key-name`**: **`operator1-keypair`**
- **`nic`**: **`net-id=production-network1`**
- **`volume`**: **`production-volume1`**

Attach the empty `production-volume2` volume to the `production-server1` instance.

4.1. Launch the `production-server1` instance using the aforementioned parameters.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--volume production-volume1 \
--key-name operator1-keypair \
--nic net-id=production-network1 \
production-server1 --wait
...output omitted...
```

4.2. Attach the empty `production-volume2` volume to the `production-server1` instance.

```
[student@workstation ~(operator1-production)]$ openstack server add \
volume production-server1 production-volume2
```

5. Access the `production-server1` instance and verify that the empty `production-volume2` volume was successfully attached to the `production-server1` instance.

5.1. Associate `172.25.250.151` as the floating IP address to the `production-server1` instance.

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

```
floating ip production-server1 172.25.250.151
```

The floating IP address `172.25.250.151` became available as a part of the lab setup.

5.2. Verify that an additional 1 GB disk representing `production-volume2` appears among the existing disks in the `production-server1` instance.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 lsblk
NAME   MAJ:MIN RM SIZE RO TYPE MOUNTPOINT
vda    253:0    0  10G  0 disk
└─vda1 253:1    0  10G  0 part /
vdb    253:16   0   1G  0 disk
```

6. Create a 1 GB partition in `production-volume2` that appears as the device `vdb` in the partition table of the `production-server1` instance. Format the partition with the `xfs` file system type and mount it temporarily on **`/mnt`**. Create the file **`/mnt/hello.txt`** with **`hello`** as the text message in it. The file **`/mnt/hello.txt`** gets accommodated into the `production-volume2` as **`/mnt`** acts as the mount point for the volume. Also, create the file **`/hello2.txt`** with **`hello2`** as the text message in it. The file **`/hello2.txt`** gets accommodated into the `production-volume1` because you are using the volume as the root file system in the `production-server1` instance.

6.1. Create a primary partition of 1 GB in the `production-volume2` that appears as the device `vdb` in the partition table of the `production-server1` instance.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 \
sudo /usr/sbin/parted /dev/vdb \
mklabel msdos \
mkpart primary 0 1G
Warning: The resulting partition is not properly aligned for best performance.
Information: You may need to update /etc/fstab.
```

NOTE

You may ignore the warning message in the output of the preceding command.

6.2. Create a file system of type `xfs` in the `/dev/vdb1` partition.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 \
sudo /usr/sbin/mkfs \
-t xfs /dev/vdb1
meta-data=/dev/vdb1              isize=512    agcount=4, agsize=65536 blks
         =                       sectsz=512   attr=2, projid32bit=1
         =                       crc=1        finobt=0, sparse=0
data     =                       bsize=4096   blocks=262143, imaxpct=25
         =                       sunit=0      swidth=0 blks
naming   =version 2              bsize=4096   ascii-ci=0 ftype=1
log      =internal log           bsize=4096   blocks=855, version=2
         =                       sectsz=512   sunit=0 blks, lazy-count=1
realtime =none                   extsz=4096   blocks=0, rtextents=0
```

6.3. Mount **`/dev/vdb1`** on **`/mnt`** temporarily.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 \
```

```
sudo /usr/bin/mount \
/dev/vdb1 /mnt
```

6.4. Create the text files **/mnt/hello.txt** and **/hello2.txt** with the text messages **hello** and **hello2** respectively.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151
[cloud-user@production-server1 ~]$ sudo -i
[root@production-server1 ~]# echo hello > /mnt/hello.txt
[root@production-server1 ~]# echo hello2 > /hello2.txt
[root@production-server1 ~]# sync
```

IMPORTANT

The **sync** command flushes contents from the Linux buffer cache to the disk. Skipping this command may cost you the contents of the text files you created.

6.5. Log off from `production-server1`.

```
[root@production-server1 ~]# logout
[cloud-user@production-server1 ~]$ logout
Connection to 172.25.250.151 closed.
[student@workstation ~(operator1-production)]$
```

7. Delete the existing `production-server1` instance and create a fresh instance with `production-volume1` as the bootable volume using the following parameters.

- **name**: **production-server1**
- **flavor**: **default**
- **key-name**: **operator1-keypair**
- **nic**: **net-id=production-network1**
- **volume**: **production-volume1**

Attach the `production-volume2` volume to the new `production-server1` instance.

7.1. Delete the existing `production-server1` instance.

```
[student@workstation ~(operator1-production)]$ openstack server delete \
production-server1
```

7.2. Create a fresh instance using the aforementioned parameters.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--volume production-volume1 \
--key-name operator1-keypair \
--nic net-id=production-network1 \
production-server1 --wait
...output omitted...
```

7.3. Attach `production-volume2` to `production-server1`.

```
[student@workstation ~(operator1-production)]$ openstack server add \
volume production-server1 production-volume2
```

8. Confirm that the text files you created earlier persist, with all of the content in the `production-server1` instance.

   8.1. Associate `172.25.250.151` as the floating IP address to the `production-server1` instance.

```
[student@workstation ~(operator1-production)]$ openstack server add \
floating ip production-server1 172.25.250.151
```

   8.2. Mount **`/dev/vdb1`** on **`/mnt`** temporarily.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 \
sudo /usr/bin/mount /dev/vdb1 /mnt
```

   8.3. Verify the contents of the **`/mnt/hello.txt`** and **`/hello2.txt`** files.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 \
cat /mnt/hello.txt
hello
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151 \
cat /hello2.txt
hello2
```

9. As `operator1`, create a 1 GB shared file system in the `production` project. Set the name of the shared file system to `production-share1`, and the type to `cephfstype`. Use the `cephfs` protocol while creating `production-share1`. For administrative operations such as managing the types of the shared file systems, use the `architect1` tenant user.

> **NOTE**
> The preinstalled Red Hat Ceph Storage in the classroom environment backs the `production-share1` shared file system. The OpenStack Shared File Systems service uses the appropriate Ceph driver to coordinate with Red Hat Ceph Storage.

   9.1. Source the `architect1` user's credentials.

```
[student@workstation ~(operator1-production)]$ source architect1-production-rc
[student@workstation ~(architect1-production)]$
```

   9.2. Verify the status of the OpenStack Shared File Systems services.

```
[student@workstation ~(architect1-production)]$ manila service-list \
--columns Binary,Status,State
+------------------+---------+-------+
| Binary           | Status  | State |
+------------------+---------+-------+
| manila-scheduler | enabled | up    |
| manila-share     | enabled | up    |
+------------------+---------+-------+
```

   9.3. Create the share type `cephfstype`. Set the value of **`driver_handles_share_servers`** to **`false`**.

```
[student@workstation ~(architect1-production)]$ manila type-create \
```

```
cephfstype false
+----------------------+-------------------------------------+
| Property             | Value                               |
+----------------------+-------------------------------------+
| required_extra_specs | driver_handles_share_servers : False |
| Name                 | cephfstype                          |
| Visibility           | public                              |
| is_default           | -                                   |
| ID                   | 384aca36-841a-4344-a75b-421e1d096e47 |
| optional_extra_specs |                                     |
| Description          | None                                |
+----------------------+-------------------------------------+
```

IMPORTANT

To dynamically scale the share nodes, set the value of **`driver_handles_share_servers`** to **`true`**.

9.4. As `operator1`, create a 1 GB shared file system called `production-share1`.

```
[student@workstation ~(architect1-production)]$ source operator1-production-rc
[student@workstation ~(operator1-production)]$ manila create \
--name production-share1 --share-type cephfstype cephfs 1
...output omitted...
```

9.5. Verify that `production-share1` was successfully created.

```
[student@workstation ~(operator1-production)]$ manila list \
--columns Name,Size,'Share Proto',Status,'Share Type Name'
+------------------+------+-------------+-----------+-----------------+
| Name             | Size | Share Proto | Status    | Share Type Name |
+------------------+------+-------------+-----------+-----------------+
| production-share1 | 1    | CEPHFS      | available | cephfstype      |
+------------------+------+-------------+-----------+-----------------+
```

10. Attach an additional NIC from the `provider-storage` network to the `production-server1` instance in order to reach the network segment of the `production-share1` shared file system from the `production-server1` instance. Use DHCP as the configuration method for the additional NIC.

10.1. Attach an additional NIC from `production-network1` to `production-server1`.

```
[student@workstation ~(operator1-production)]$ openstack server add \
network production-server1 provider-storage
```

10.2. Open an SSH session to `production-server1` and confirm the existence of the additional NIC in the instance.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151
[cloud-user@production-server1 ~]$ ip a s
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
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1442 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether fa:16:3e:b1:87:d9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.11/24 brd 192.168.1.255 scope global noprefixroute dynamic eth0
       valid_lft 42295sec preferred_lft 42295sec
    inet6 fe80::f816:3eff:feb1:87d9/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether fa:16:3e:2e:e2:2d brd ff:ff:ff:ff:ff:ff
```

Notice the `eth1` network interface that you added in the preceding step.

10.3. Create the **`/etc/sysconfig/network-scripts/ifcfg-eth1`** file and set DHCP as the configuration method for `eth1`. Use the **sudo** command when creating the interface file because this operation requires root privileges.

```
DEVICE=eth1
ONBOOT=yes
BOOTPROTO=dhcp
```

10.4. Bring up the `eth1` interface.

```
[cloud-user@production-server1 ~]$ sudo ifup eth1
```

10.5. Verify the settings of `eth1`.

```
[cloud-user@production-server1 ~]$ ip a s
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
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1442 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether fa:16:3e:b1:87:d9 brd ff:ff:ff:ff:ff:ff
    inet 192.168.1.11/24 brd 192.168.1.255 scope global noprefixroute dynamic eth0
       valid_lft 41615sec preferred_lft 41615sec
    inet6 fe80::f816:3eff:feb1:87d9/64 scope link
       valid_lft forever preferred_lft forever
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
    link/ether fa:16:3e:2e:e2:2d brd ff:ff:ff:ff:ff:ff
    inet 172.24.3.205/24 brd 172.24.3.255 scope global noprefixroute dynamic eth1
       valid_lft 43100sec preferred_lft 43100sec
    inet6 fe80::f816:3eff:fe2e:e22d/64 scope link
```

```
        valid_lft forever preferred_lft forever
```

Notice the `172.24.3.205` IP address of the `eth1` interface. This IP address allows you to reach the network of `production-share1` shared file system from the `production-server1` instance.

10.6. Log off from `production-server1`.

```
[cloud-user@production-server1 ~]$ logout
Connection to 172.25.250.151 closed.
[student@workstation ~(operator1-production)]$
```

11. Create the cephx user `client.cloud-user` and save the key ring as **`cloud-user.keyring`**. Authenticate as `client.manila` using the key ring from **`/etc/ceph/ceph.client.manila.keyring`** while creating `client.cloud-user`. Transfer the appropriate Ceph configuration file and the **`cloud-user.keyring`** file to `production-server1`. Install any required utilities on `production-server1` to access the Ceph-backed shared file system using the repository configuration file from `http://materials.example.com/ceph.repo`.

11.1. Open an SSH session to `controller0` that acts as the Ceph Monitor and has all the required administrative tools.

```
[student@workstation ~(operator1-production)]$ ssh root@controller0
[root@controller0 ~]#
```

11.2. Create the cephx user `client.cloud-user` and save its key ring to **`/root/cloud-user.keyring`**. Authenticate as `continual` using the key ring from **`/etc/ceph/ceph.client.manila.keyring`** while creating `canticles`.

```
[root@controller0 ~]# ceph --name=client.manila \
--keyring=/etc/ceph/ceph.client.manila.keyring \
auth get-or-create client.cloud-user > /root/cloud-user.keyring
```

11.3. Log off from `controller0`.

```
[root@controller0 ~]# logout
Connection to controller0 closed.
[student@workstation ~(operator1-production)]$
```

11.4. Transfer the appropriate Ceph configuration file and the **`cloud-user.keyring`** file to `production-server1`.

```
[student@workstation ~(operator1-production)]$ scp \
root@controller0:{/etc/ceph/ceph.conf,/root/cloud-user.keyring} manila/
ceph.conf                                     100%  797   870.4KB/s   00:00
cloud-user.keyring                            100%   68    75.9KB/s   00:00
[student@workstation ~(operator1-production)]$ scp \
manila/{ceph.conf,cloud-user.keyring} cloud-user@172.25.250.151:
ceph.conf                                     100%  797   670.0KB/s   00:00
cloud-user.keyring                            100%   68    80.6KB/s   00:00
```

11.5. Install *ceph-fuse* on the `production-server1` instance. Download the repository configuration file from `http://materials.example.com/ceph.repo`.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151
```

```
[cloud-user@production-server1 ~]$ sudo curl -s -f -o \
/etc/yum.repos.d/ceph.repo http://materials.example.com/ceph.repo
[cloud-user@production-server1 ~]$ sudo yum install ceph-fuse
...output omitted...
Total download size: 3.3 M
Installed size: 11 M
Is this ok [y/d/N]: y
...output omitted...
Complete!
```

11.6. Log off from `production-server1`.

```
[cloud-user@production-server1 ~]$ logout
[student@workstation ~(operator1-production)]$
```

12. Grant **`read`** and **`write`** permissions on `production-share1` to `client.cloud-user`.

12.1. Grant **`read`** and **`write`** permissions on `production-share1` to `client.cloud-user`.

```
[student@workstation ~(operator1-production)]$ manila access-allow \
--access-level rw \
production-share1 cephx cloud-user
+--------------+--------------------------------------+
| Property     | Value                                |
+--------------+--------------------------------------+
| access_key   | None                                 |
| share_id     | 84ce9714-2093-436b-ae70-4bd7fe48fa00 |
| created_at   | 2018-10-31T19:27:40.000000           |
| updated_at   | None                                 |
| access_type  | cephx                                |
| access_to    | cloud-user                           |
| access_level | rw                                   |
| state        | queued_to_apply                      |
| id           | ea602cdc-b303-4966-9a23-8fa07aad9969 |
+--------------+--------------------------------------+
```

12.2. Verify the access permissions on `production-share1`.

```
[student@workstation ~(operator1-production)]$ manila access-list \
--columns access_type,access_to,access_level,state \
production-share1
+-------------+------------+--------------+--------+
| Access_Type | Access_To  | Access_Level | State  |
+-------------+------------+--------------+--------+
| cephx       | cloud-user | rw           | active |
+-------------+------------+--------------+--------+
```

13. Access the `production-share1` shared file system from the `production-server1` instance. Authenticate as `client.cloud-user` using the key ring from **`/home/cloud-user/cloud-user.keyring`** while accessing `production-share1`. The appropriate

Ceph cluster settings are in **/home/cloud-user/ceph.conf**. Create the file **hello.txt** with **hello** as the text message in the shared file system.

13.1. Open an SSH session to `production-server1` and switch to `root`.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151
[cloud-user@production-server1 ~]$ sudo -i
[root@production-server1 ~]#
```

13.2. Create **/manila** as the mount point for `production-share1`.

```
[root@production-server1 ~]# mkdir /manila
```

13.3. Open a new terminal on `workstation` as `student` and determine the export location of `production-share1` as the `operator1` tenant user.

```
[student@workstation ~]$ source operator1-production-rc
[student@workstation ~(operator1-production)]$ manila share-export-location-list \
production-share1 --columns Path
+-----------------------------------------------------------------------+
| Path                                                                  |
+-----------------------------------------------------------------------+
| 172.24.3.1:6789:/volumes/_nogroup/45306f38-73c9-44b3-a794-a84bdb0b25e3 |
+-----------------------------------------------------------------------+
```

Take note of the export location because you are going to use it while mounting the `production-share1` shared file system in the following step.

13.4. Return to the first terminal and mount `production-share1` on **/manila** from `production-server1`. Authenticate as `cloud-user.keyring` using **/home/cloud-user/cloud-user.keyring** while mounting. Use **/home/cloud-user/ceph.conf** for the appropriate Ceph cluster settings. Use the same export location as in the preceding output.

```
[root@production-server1 ~]# ceph-fuse /manila \
--id=cloud-user --conf=/home/cloud-user/ceph.conf \
--keyring=/home/cloud-user/cloud-user.keyring \
--client-mountpoint=/volumes/_nogroup/45306f38-73c9-44b3-a794-a84bdb0b25e3
2018-10-31 16:30:39.637732 7f8163dc00c0 -1 init, newargv = 0x55855fcead80
 newargc=9
ceph-fuse[1877]: starting ceph client
ceph-fuse[1877]: starting fuse
```

13.5. Verify that `production-share1` is successfully mounted.

```
[root@production-server1 ~]# df -Th
Filesystem     Type            Size  Used Avail Use% Mounted on
/dev/vda1      xfs              10G  1.6G  8.5G  16% /
devtmpfs       devtmpfs        898M     0  898M   0% /dev
tmpfs          tmpfs           920M     0  920M   0% /dev/shm
tmpfs          tmpfs           920M   17M  903M   2% /run
tmpfs          tmpfs           920M     0  920M   0% /sys/fs/cgroup
tmpfs          tmpfs           184M     0  184M   0% /run/user/1001
```

```
ceph-fuse      fuse.ceph-fuse  1.0G     0  1.0G   0% /manila
```

13.6. Create **/manila/hello.txt** with **hello** as the text message in it.

```
[root@production-server1 ~]# echo hello > /manila/hello.txt
```

13.7. Unmount `production-share1` from **/manila**.

```
[root@production-server1 ~]# umount /manila
```

13.8. Log off from `production-server1`.

```
[root@production-server1 ~]# logout
[cloud-user@production-server1 ~]$ logout
Connection to 172.25.250.151 closed.
[student@workstation ~(operator1-production)]$
```

14. Delete the `production-server1` instance and create a new instance with the following parameters:

- **name**: **production-server2**
- **flavor**: **default**
- **key-name**: **operator1-keypair**
- **nic**: **net-id=production-network1**
- **nic**: **net-id=provider-storage**
- **volume**: **production-volume1**
- **user-data**: **/home/student/manila/user-data.file**

> **NOTE**
> The user data file **/home/student/manila/user-data.file** contributes to configuring the second NIC of the instance.

Access the `production-share1` shared file system from the new `production-server2` instance to confirm that the text file you created in the shared file system in a previous step persists.

14.1. Delete the `production-server1` instance.

```
[student@workstation ~(operator1-production)]$ openstack server delete \
production-server1
```

14.2. Create the new `production-server2` instance with the aforementioned parameters.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--volume production-volume1 \
--key-name operator1-keypair \
--nic net-id=production-network1 \
--nic net-id=provider-storage \
--user-data /home/student/manila/user-data.file \
production-server2 --wait
```

```
...output omitted...
```

14.3. Associate the `172.25.250.151` floating IP address with `production-server2`.

```
[student@workstation ~(operator1-production)]$ openstack server add \
floating ip production-server2 172.25.250.151
```

14.4. Open an SSH session to `production-server2` and switch to `root`.

```
[student@workstation ~(operator1-production)]$ ssh 172.25.250.151
[cloud-user@production-server2 ~]$ sudo -i
[root@production-server2 ~]#
```

14.5. Mount `production-share1` on **`/manila`** from `production-server2`. Authenticate as `cloud-user.keyring` using **`/home/cloud-user/cloud-user.keyring`** while mounting. Use **`/home/cloud-user/ceph.conf`** for the appropriate Ceph cluster settings. Use the same export location as in the preceding steps.

```
[root@production-server2 ~]# ceph-fuse /manila \
--id=cloud-user --conf=/home/cloud-user/ceph.conf \
--keyring=/home/cloud-user/cloud-user.keyring \
--client-mountpoint=/volumes/_nogroup/45306f38-73c9-44b3-a794-a84bdb0b25e3
ph-fuse[1377]: starting ceph client
2018-11-01 02:18:25.304052 7f59130e70c0 -1 init, newargv = 0x55c9235b6d80
 newargc=9
ceph-fuse[1377]: starting fuse
```

14.6. Verify that `production-share1` is successfully mounted.

```
[root@production-server2 ~]# df -Th
Filesystem     Type            Size  Used Avail Use% Mounted on
/dev/vda1      xfs              10G  1.6G  8.5G  16% /
devtmpfs       devtmpfs        898M     0  898M   0% /dev
tmpfs          tmpfs           920M     0  920M   0% /dev/shm
tmpfs          tmpfs           920M   17M  903M   2% /run
tmpfs          tmpfs           920M     0  920M   0% /sys/fs/cgroup
tmpfs          tmpfs           184M     0  184M   0% /run/user/1001
ceph-fuse      fuse.ceph-fuse  1.0G     0  1.0G   0% /manila
```

14.7. Confirm that the text file you created in the earlier steps in the `production-share1` shared file system still persists with all of its content.

```
[root@production-server2 ~]# cat /manila/hello.txt
hello
```

14.8. Log off from `production-server2` and delete the instance.

```
[root@production-server2 ~]# logout
[cloud-user@production-server2 ~]$ logout
Connection to 172.25.250.151 closed.
[student@workstation ~(operator1-production)]$ openstack server delete \
production-server2
```

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab storage-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The block-based, object-based, and file-based storage options have their individual applicability and use cases. These storage options do not act as replacements or alternatives for each other. The block-based storage option offers more reliability, the object-based storage offers better scalability, and the file-based storage extends simplicity to the tenant users.
- Red Hat OpenStack Platform supports both Red Hat Ceph Storage and NFS as storage back ends.
- The Red Hat Ceph Storage architecture is based on monitor (MON) daemons and object storage device (OSD) daemons.
- Red Hat Ceph Storage features include seamless scalability and no single point of failure.
- Red Hat Ceph Storage uses the CephX authentication mechanism to identify and authorize users.
- The Red Hat OpenStack Platform block storage and image services use RBDs to access Ceph, and require both a user and pool to access the cluster.
- The Red Hat OpenStack Platform object storage service (Swift) provides scalable object storage to the tenant users.
- The OpenStack object store architecture includes a front-end service, the proxy server, and three back-end services: the account server, the object server, and the container server.
- Users can create containers in the object store, and upload objects to those containers.
- The OpenStack Shared File Systems service manages the shares that serve the file-based storage requirements. Users from the instances access the shares as remote file systems. The shares can be hosted on various back-end storage providers including, but not limited to, Red Hat Ceph Storage and Red Hat Gluster Storage.
- The data in the ephemeral storage does not persist after instance termination, in contrast to the persistent storage that stores data persistently. The persistent storage mainly stores data intended to be reusable.

CHAPTER 6

MANAGING OPENSTACK NETWORKING

GOAL

Manage and troubleshoot virtual network infrastructure.

OBJECTIVES

- Explain the different network types available to the OpenStack networking service.
- Improve network performance with Open Virtual Networking.
- Compare the provisioning options for self-service and provider networks, and compare centralized and distributed networks.

SECTIONS

- Describing Network Protocol Types (and Guided Exercise)
- Introducing Open Virtual Networking (OVN) (and Guided Exercise)
- Selecting Network Provisioning Options (and Guided Exercise)

LAB

- Managing OpenStack Networking

DESCRIBING NETWORK PROTOCOL TYPES

OBJECTIVES

After completing this section, students should be able to:

- Discuss *Software-defined networking (SDN).*
- Discuss SDN implementation and use cases.
- Discuss flat, VLAN, and Geneve network types.

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.

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 endpoint requirements, or how fast 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.

Benefits of SDN

Consumers continue to demand fast, reliable, secure, and omnipresent network connections. Service providers are utilizing virtualization and SDN technologies to better meet those needs.

Benefits of SDN include:

- The decoupling of the control plane and data plane enables both planes to evolve independently, which results in several advantages such as high flexibility, being vendor-agnostic, open programmability, and a centralized network view.
- SDN adds security features that allow administrators to route traffic through a single, centrally located, firewall. One advantage of this is the ability to utilize intrusion detection methods on real-time captures of network traffic.
- Automated load balancing in SDNs enhances the performance of server load balancing, and reduces the complexity of implementation.
- Network scalability allows data centers to use features of software-defined networking along with virtualized servers and storage to implement dynamic environments where computing resources are added and removed as needed.
- SDN reduces operational costs by minimizing the need to deploy, maintain, and replace expensive hardware such as many of the servers and network switches within a data center.

INTRODUCING THE SDN ARCHITECTURE

SDN is based on the concept of separation between controlled services and the controllers that control those services. Controllers manipulate services by way of interfaces. Interfaces are mainly

API invocations through a library or system call. However, such interfaces may be extended with protocol definitions that use local *interprocess communication (IPC)* or a protocol that can also act remotely. A protocol may be defined as an open standard or in a proprietary manner.

Architectural Components

The following list defines and explains the architectural components:

Application Plane
The plane where applications and services that define network behavior reside.

Management Plane
Handles monitoring, configuration, and maintenance of network devices, such as making decisions regarding the state of a network device.

Control Plane
Responsible for making decisions on how packets should be forwarded by one or more network devices, and for pushing such decisions down to the network devices for execution.

Operational Plane
Responsible for managing the operational state of the network device, such as whether the device is active or inactive, the number of ports available, the status of each port, and so on.

Forwarding Plane
Responsible for handling packets in the data path based on the instructions received from the control plane. Actions of the forwarding plane include forwarding, dropping, and changing packets.

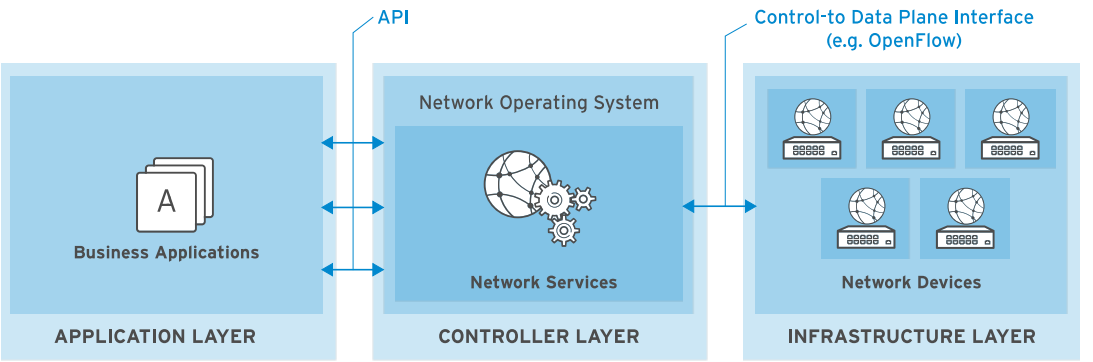


Figure 6.1: The SDN architecture

SDN Terminology

Application
SDN applications are programs that communicate their network requirements and desired network behavior to the SDN controller over a *northbound interface (NBI)*.

Data path
The SDN data path is a logical network device that exposes visibility control over its advertised forwarding and data processing capabilities. An SDN data path comprises a *Control to Data Plane Interface (CDPI)* agent and a set of one or more traffic forwarding engines.

Controller
The SDN controller is a logically centralized entity in charge of translating the requirements from the SDN application layer down to the SDN data paths. SDN controllers provide a view of the network to the SDN applications.

Control to Data Plane Interface (CDPI)
The CDPI is the interface defined between an SDN controller and an SDN data path that provides control of all forwarding operations, capabilities advertisement, statistics reporting, and event notification.

Northbound Interfaces (NBIs)
NBIs are interfaces between SDN applications and SDN controllers. They typically provide network views and enable expression of network behavior and requirements.

SDN Classroom Components

On the `controller0` node, use the **`ps -ef`** command to list the OVN processes. Notice the two running processes, `ovsdb-server` and `ovn-northd`. In the first output, notice that the log files, the pid files and ssl-protocols all point to Northbound components. In the second output, the elements for `ovsdb-server` all point to Southbound components. `ovn-northd` shows up in both outputs as it is connected to both Northbound and Southbound databases.

```
[root@controller0 ~]# ps -ef | grep ovnnb
root       28393   28392  0 Oct07 ?        00:00:05 ovsdb-server --detach --
monitor -vconsole:off -vfile:info --log-file=/var/log/openvswitch/ovsdb-server-
nb.log --remote=punix:/var/run/openvswitch/ovnnb_db.sock --pidfile=/var/run/
openvswitch/ovnnb_db.pid --remote=db:OVN_Northbound,NB_Global,connections
 --unixctl=ovnnb_db.ctl --private-key=db:OVN_Northbound,SSL,private_key
 --certificate=db:OVN_Northbound,SSL,certificate
 --ca-cert=db:OVN_Northbound,SSL,ca_cert --ssl-
protocols=db:OVN_Northbound,SSL,ssl_protocols --ssl-
ciphers=db:OVN_Northbound,SSL,ssl_ciphers --remote=ptcp:6641:172.24.1.50 --sync-
from=tcp:192.0.2.254:6641 /etc/openvswitch/ovnnb_db.db
root       28840   28839  0 Oct07 ?        00:00:00 ovn-northd -vconsole:emer
 -vsyslog:err -vfile:info --ovnnb-db=unix:/var/run/openvswitch/ovnnb_db.sock --
ovnsb-db=unix:/var/run/openvswitch/ovnsb_db.sock --no-chdir --log-file=/var/log/
openvswitch/ovn-northd.log --pidfile=/var/run/openvswitch/ovn-northd.pid --detach
 --monitor
```

```
[root@controller0 ~]# ps -ef | grep ovnsb
root       28403   28402  0 Oct07 ?        00:00:19 ovsdb-server --detach --
monitor -vconsole:off -vfile:info --log-file=/var/log/openvswitch/ovsdb-server-
sb.log --remote=punix:/var/run/openvswitch/ovnsb_db.sock --pidfile=/var/run/
openvswitch/ovnsb_db.pid --remote=db:OVN_Southbound,SB_Global,connections
 --unixctl=ovnsb_db.ctl --private-key=db:OVN_Southbound,SSL,private_key
 --certificate=db:OVN_Southbound,SSL,certificate
 --ca-cert=db:OVN_Southbound,SSL,ca_cert --ssl-
protocols=db:OVN_Southbound,SSL,ssl_protocols --ssl-
ciphers=db:OVN_Southbound,SSL,ssl_ciphers --remote=ptcp:6642:172.24.1.50 --sync-
from=tcp:192.0.2.254:6642 /etc/openvswitch/ovnsb_db.db
root       28840   28839  0 Oct07 ?        00:00:00 ovn-northd -vconsole:emer
 -vsyslog:err -vfile:info --ovnnb-db=unix:/var/run/openvswitch/ovnnb_db.sock --
ovnsb-db=unix:/var/run/openvswitch/ovnsb_db.sock --no-chdir --log-file=/var/log/
openvswitch/ovn-northd.log --pidfile=/var/run/openvswitch/ovn-northd.pid --detach
 --monitor
```

In **`/var/log/openvswitch`** there are two original log files, **`ovsdb-server.log`** and **`ovs-vswitchd.log`**.

```
[root@controller0 ~]# view ovsdb-server.log
```

```
2018-10-08T03:44:01.297Z|00008|vlog|INFO|opened log file /var/log/openvswitch/
ovsdb-server.log
[root@controller0 ~]# view ovs-vswitchd.log
2018-10-08T03:44:01.300Z|00058|vlog|INFO|opened log file /var/log/openvswitch/ovs-
vswitchd.log
2018-10-08T09:15:40.514Z|00059|bridge|INFO|bridge br-ex: added interface patch-
provnet-4f582157-2f9f-4e4b-b542-2aed6e75ea7a-to-br-int on port 2
2018-10-08T09:15:40.514Z|00060|bridge|INFO|bridge br-int: added interface patch-
br-int-to-provnet-4f582157-2f9f-4e4b-b542-2aed6e75ea7a on port 5
2018-10-08T09:15:50.511Z|00061|connmgr|INFO|br-int<->unix#0: 229 flow_mods 10 s
 ago (227 adds, 2 modifications)
2018-10-08T09:17:36.274Z|00062|connmgr|INFO|br-int<->unix#0: 40 flow_mods 10 s ago
 (39 adds, 1 deletes)
2018-10-08T09:18:36.273Z|00063|connmgr|INFO|br-int<->unix#0: 4 flow_mods in the 21
 s starting 24 s ago (3 adds, 1 modifications)
2018-10-08T09:22:27.691Z|00064|connmgr|INFO|br-int<->unix#0: 2 flow_mods 10 s ago
 (2 adds)
2018-10-08T09:23:27.691Z|00065|connmgr|INFO|br-int<->unix#0: 3 flow_mods 27 s ago
 (2 adds, 1 modifications)
2018-10-08T11:20:30.844Z|00066|connmgr|INFO|br-int<->unix#0: 43 flow_mods 10 s ago
 (43 adds)
2018-10-08T11:21:30.844Z|00067|connmgr|INFO|br-int<->unix#0: 1 flow_mods 50 s ago
 (1 adds)
```

The **`readme.txt`** file points to the directory where the container logs are stored. Note that the log files in this directory are different from those in **`/var/log/openvswitch/`**.

```
[root@controller0 ~]# view readme.txt
Log files from openvswitch containers can be found under
/var/log/containers/openvswitch.
```

```
[root@controller0 ~]# ll /var/log/containers/openvswitch
-rw-r-----. 1 root root   97 Oct  9 03:16 ovn-controller.log
-rw-r-----. 1 root root  410 Oct  9 03:16 ovn-controller.log.1
-rw-r-----. 1 root root  142 Oct  8 03:44 ovn-controller.log.2.gz
-rw-r-----. 1 root root 1098 Oct  7 15:16 ovn-controller.log.3.gz
-rw-r-----. 1 root root 1204 Oct  7 14:17 ovn-northd.log.1
-rw-r-----. 1 root root 6111 Oct  7 14:17 ovsdb-server-nb.log.1
-rw-r-----. 1 root root 5768 Oct  7 14:17 ovsdb-server-sb.log.1
```

```
[root@controller0 ~]# view ovn-controller.log
2018-10-08T03:44:01.293Z|00064|vlog|INFO|opened log file /var/log/openvswitch/ovn-
controller.log
2018-10-08T09:15:40.478Z|00065|binding|INFO|Claiming lport cr-lrp-724e47af-
a7cc-4e6f-8e35-3b530494824a for this chassis.
2018-10-08T09:15:40.478Z|00066|binding|INFO|cr-lrp-724e47af-
a7cc-4e6f-8e35-3b530494824a: Claiming fa:16:3e:fa:ff:0e 172.25.250.103/24
```

```
[root@controller0 ~]# view ovn-northd.log
2018-10-06T08:33:29.492Z|00001|vlog|INFO|opened log file /var/log/openvswitch/ovn-
northd.log
2018-10-06T08:33:29.493Z|00002|reconnect|INFO|unix:/var/run/openvswitch/
ovnnb_db.sock: connecting...
```

```
2018-10-06T08:33:29.494Z|00003|reconnect|INFO|unix:/var/run/openvswitch/
ovnsb_db.sock: connecting...
2018-10-06T08:33:29.494Z|00004|reconnect|INFO|unix:/var/run/openvswitch/
ovnnb_db.sock: connected
2018-10-06T08:33:29.494Z|00005|reconnect|INFO|unix:/var/run/openvswitch/
ovnsb_db.sock: connected
2018-10-06T08:33:29.497Z|00006|ovn_northd|INFO|ovn-northd lock acquired. This ovn-
northd instance is now active.
2018-10-07T14:17:39.088Z|00001|vlog|INFO|opened log file /var/log/openvswitch/ovn-
northd.log
2018-10-07T14:17:39.094Z|00002|reconnect|INFO|unix:/var/run/openvswitch/
ovnnb_db.sock: connecting...
2018-10-07T14:17:39.094Z|00003|reconnect|INFO|unix:/var/run/openvswitch/
ovnsb_db.sock: connecting...
2018-10-07T14:17:39.094Z|00004|reconnect|INFO|unix:/var/run/openvswitch/
ovnnb_db.sock: connected
2018-10-07T14:17:39.094Z|00005|reconnect|INFO|unix:/var/run/openvswitch/
ovnsb_db.sock: connected
2018-10-07T14:17:39.096Z|00006|ovn_northd|INFO|ovn-northd lock acquired. This ovn-
northd instance is now active.
```

On `compute0` use the **ps** command to list the OVN processes. Notice that the only OVN processes are metadata agents.

```
[root@compute0 ~]# ps -ef | grep ovn
42435       9438    9398  0 Oct07 ?        00:01:03 /usr/bin/python2 /usr/bin/
networking-ovn-metadata-agent --config-file /etc/neutron/neutron.conf --config-
file /etc/neutron/plugins/networking-ovn/networking-ovn-metadata-agent.ini --
config-dir /etc/neutron/conf.d/networking-ovn-metadata-agent --log-file=/var/log/
neutron/ovn-metadata-agent.log
root        9475    9452  0 Oct07 ?        00:00:44 /usr/bin/ovn-controller --
pidfile --log-file unix:/run/openvswitch/db.sock
42435      10690    9438  0 Oct07 ?        00:00:01 /usr/bin/python2 /usr/bin/
networking-ovn-metadata-agent --config-file /etc/neutron/neutron.conf --config-
file /etc/neutron/plugins/networking-ovn/networking-ovn-metadata-agent.ini --
config-dir /etc/neutron/conf.d/networking-ovn-metadata-agent --log-file=/var/log/
neutron/ovn-metadata-agent.log
...output omitted...
42435     196986    9438  0 09:18 ?        00:00:00 haproxy -f /var/lib/neutron/
ovn-metadata-proxy/52653d3a-8b06-414e-9155-b658d06a4f52.conf
```

On the `computehci0` node, use the **ps** command to list the OVN processes. Note that the HCI node has the `ovn-controller` and `metadata agent` processes.

```
[root@computehci0 ~]# ps -ef | grep ovn
root       10451   10373  0 Oct07 ?        00:00:06 /usr/bin/ovn-controller --
pidfile --log-file unix:/run/openvswitch/db.sock
42435      10651   10625  0 Oct07 ?        00:01:10 /usr/bin/python2 /usr/bin/
networking-ovn-metadata-agent --config-file /etc/neutron/neutron.conf --config-
file /etc/neutron/plugins/networking-ovn/networking-ovn-metadata-agent.ini --
config-dir /etc/neutron/conf.d/networking-ovn-metadata-agent --log-file=/var/log/
neutron/ovn-metadata-agent.log
42435      13085   10651  0 Oct07 ?        00:00:01 /usr/bin/python2 /usr/bin/
networking-ovn-metadata-agent --config-file /etc/neutron/neutron.conf --config-
```

```
file /etc/neutron/plugins/networking-ovn/networking-ovn-metadata-agent.ini --
config-dir /etc/neutron/conf.d/networking-ovn-metadata-agent --log-file=/var/log/
neutron/ovn-metadata-agent.log
```

INTRODUCING NETWORK TYPES

Administrators should be familiar with networking concepts when working with Red Hat OpenStack Platform. The OpenStack Networking service is the SDN networking project that provides *Networking-as-a-Service (NaaS)* in virtual environments. It implements traditional networking features such as subnetting, bridging, VLANs, and more recent technologies, such as *Open Virtual Network (OVN)*. Open Virtual Network is the default SDN for Red Hat OpenStack Platform.

Flat Networks

To connect instances directly to the external network use flat provider networks. Flat networks are useful if there are multiple physical networks and separate physical interfaces. This option does not require a switch that performs VLAN tagging. Flat networks can only be configured by administrators. All tenant networks can use a flat provider network. A flat network does not support segmentation.

Virtual LAN (VLAN) Networks

A local area network (LAN) is composed of an Ethernet segment enabling direct Layer 2 MAC access between computer nodes. Communicating between computer nodes on different LANs requires router devices that forward packets between segments, through ports controlled by Layer 3 routing tables.

Virtual LAN (VLAN) functionality adds further segregation and Ethernet segment partitioning by including optional packet-based VLAN identifiers (tags). These identifiers are defined in the IEEE 802.1Q standard, referred to as "Dot1Q". A 32-bit field, placed in the Ethernet frame header, includes a Tag Protocol Identifier (TPID). The value of that 32-bit field is 0x8100. It includes a VLAN Identifier field indicating the virtual segment to which this frame belongs. Excluding two reserved values, the 12-bit size of the VLAN Identifier allows a maximum of 4094 unique VLANs on a single LAN segment.

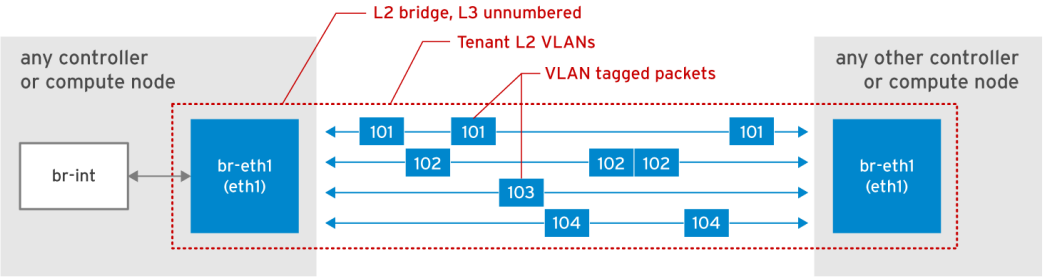


Figure 6.2: VLAN tagging on a Linux bridge

All VLAN operations in OpenStack use 802.1Q tagging, but tagged VLAN segments are limited in range. VLAN boundaries are defined by specific Open vSwitch bridges where VLAN functionality is configured and translated for each network type.

Introduction to Geneve

VLANs were created to segment network traffic. Unfortunately, their 12-bit field limited the number of VLANs to around 4000. Before the cloud era this limited number was more than enough.

VXLAN (Virtual Extensible LAN) addressed the limitations of VLAN. The larger 24-bit header size ensured a greater number of virtual networks. VXLAN did not require any change to the hardware infrastructure but it was incompatible with other networking solutions, such as STT and NVGRE.

Geneve (Generic Network Virtualization Encapsulation) addresses the limitations by supporting all of the capabilities of VXLAN, NVGRE and STT. Geneve only encapsulates the data format, meaning that it does not include any specification for the control plane. The Geneve protocol evolves by adding new optional capabilities, which are transmitted via standard networking equipment without the need for agents. Packets are sent using unicast or multicast addressing to tunnel endpoints. Packets are transmitted to the endpoints using a standard UDP packet, supported by IPv4 and IPv6.

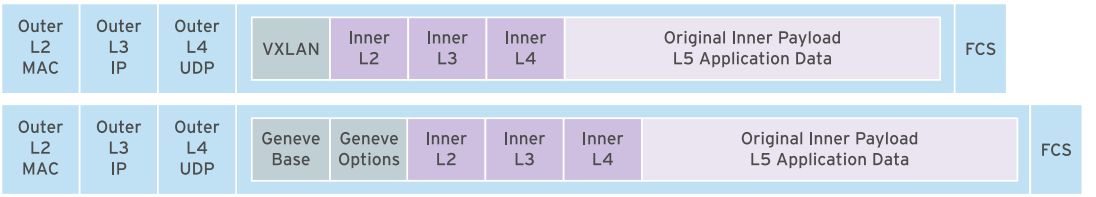


Figure 6.3: VXLAN and Geneve frame comparison

Geneve frame details

HEADER	PERTINENT FIELDS	COMMENT
Outer L2	Destination MAC Address	Normal hop-to-hop L2 header.
	Source MAC Address	
	EtherType 0x0800 (IPv4) or IPv6 (0x86dd)	
Outer L3	Protocol 0x11 (UDP)	The source and destination addresses are VTEP endpoints on the OVS tunnel bridge (br-tun).
	Source IP Address	
	Destination IP Address	
Outer L4	UDP Source Port (Geneve 6801)	The VTEP service port.
Geneve Base Header	Version, flags and reserved bits	The protocol type is for the original frame. The 24-bit VNI field allows 16 million possible tunnel identifiers.
	Protocol Type	
	Virtual network identifier (VNI)	
Geneve Options	Class, type, reserved, length	Options are chainable extensions, in type, length, value (TLV) format.
	Veriable length options	
Original Frame	L2, L3, L4 Headers	Carried as payload, unmodified during transport.
	L5 Application Data	

Comparing Geneve to VXLAN and GRE

There are a number of existing protocols as previously mentioned, for example GRE, VXLAN and *STT (Stateless Tunneling Transport)*. So why is Geneve the default choice for OVN?

The existing protocols lack extensibility to allow for data plane innovation. The control plane and data plane are two unrelated components, but protocols such as GRE and VXLAN link them together. This architecture does not easily allow for the modification or modernization of either component. Metadata is the most important element in extensibility. The VXLAN protocol can only include the 24-bit VNI and a few flags in its header. The GRE header also has only a 24 bit ID plus flags in its Key field. STT allows 64 bits of context information in additon to a 12-bit VLAN ID, but is not extensible.

With previous protocol fixed header sizes, the most that can be achieved is network isolation or partitioning. A tunnel protocol should provide additional services, such as security, telemetry, encryption and authentication. However, each additional service requires additional space in the protocol header. This is where VXLAN and GRE are limited.

The encapsulation protocol should not limit what services can be managed by the virtualized network. An extra extension to VXLAN could be one method to resolve this problem, but that would mean changing the existing protocol.

Geneve solves the space problem by allowing optional fields to extend the metadata. This extensible encapsulation format allows for future innovation. It decouples the control plane and data plane components, thus allowing for different rates of development. It continues to use standard IP as the underlay and supports multiple encapsulated protocols.

The Geneve header format is based on UDP. It is very similar to the VXLAN header and provides favorable transport over existing IP networks. A base small fixed header was added to include some of the properties of the other protocols. A set of *Type Length Value (TLV)* options were then added. TLVs are used in many protocols to allow for extensibility.

A key element of OVN is the use of logical flows. Development is easier when using logical flows because the physical location of a resource is irrelevant. Logical flows do not have to be written for each switch; the entire deployment is regarded as one switch. To implement this design, a network ID is passed over a tunnel along with IDs for the logical source and destination ports. In Geneve, the network is identified using the VNI field. The extra IDs for the logical source and destination ports are implemented by using an additional 32-bit TLV. This is not possible in VXLAN or GRE.

The Geneve header includes a 24-bit *Datapath ID*, a 15-bit ingress port, and a 16-bit egress port. With the addition of the ingress and egress ports OVN can implement OpenStack security group rules, something that is not possible in VXLAN or GRE.

Geneve Components

Geneve traffic can be seen on the tenant network `VLAN20`. Like VXLAN it has a unique VNI. In this example the tunnel ID is *0x3*, `172.24.2.2.35614` corresponds to `compute0` and `172.24.2.12.6081` corresponds to `compute1`. The IP addresses in the *192.168.1.X* subnet represent two instances on the same OpenStack network.

```
8e:43:4c:8b:d8:cf > 76:3a:02:69:82:1e, ethertype IPv4 (0x0800), length
 156: 172.24.2.2.35614 > 172.24.2.12.6081: Geneve, Flags [C], vni 0x3, proto TEB
 (0x6558), options [8 bytes]: fa:16:3e:b5:fd:da > fa:16:3e:ec:5f:e0, ethertype
 IPv4 (0x0800), length 98: 192.168.1.11 > 192.168.1.12: ICMP echo request, id
 1211, seq 1, length 64

   76:3a:02:69:82:1e > 8e:43:4c:8b:d8:cf, ethertype IPv4 (0x0800), length
 156: 172.24.2.12.23252 > 172.24.2.2.6081: Geneve, Flags [C], vni 0x3, proto TEB
 (0x6558), options [8 bytes]: fa:16:3e:ec:5f:e0 > fa:16:3e:b5:fd:da, ethertype
 IPv4 (0x0800), length 98: 192.168.1.12 > 192.168.1.11: ICMP echo reply, id 1211,
 seq 1, length 64
```

The Geneve port interfaces can be found on the overcloud nodes. For example, on `controller0` using the **`ovs-vsctl`** command list the configuration of br-int. Note the different `ovn-` ports. The IP addresses match the three compute nodes in the classroom environment. The `patch-br-int` port is the classroom provider network `provider-datacentre`.

```
[root@controller0 ~]# ovs-vsctl show
Bridge br-int
    fail_mode: secure
    Port "ovn-840e32-0"
         Interface "ovn-840e32-0"
         type: geneve
         options: {csum="true", key=flow, remote_ip="172.24.2.2"}
    Port "ovn-14c844-0"
         Interface "ovn-14c844-0"
         type: geneve
         options: {csum="true", key=flow, remote_ip="172.24.2.12"}
    Port "o-hm0"
         Interface "o-hm0"
         type: internal
    Port "ovn-263d95-0"
         Interface "ovn-263d95-0"
         type: geneve
         options: {csum="true", key=flow, remote_ip="172.24.2.6"}
    Port br-int
         Interface br-int
         type: internal
    Port "patch-br-int-to-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00"
         Interface "patch-br-int-to-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00"
         type: patch
         options: {peer="patch-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00-to-br-
int"}
  ovs_version: "2.9.0"
...output omitted...
```

```
[student@workstation ~(architect1-finance)]$ openstack hypervisor list \
-c "Hypervisor Hostname" -c "Host IP"
+-----------------------------------+-------------+
| Hypervisor Hostname               | Host IP     |
+-----------------------------------+-------------+
| computehci0.overcloud.example.com | 172.24.1.6  |
| compute1.overcloud.example.com    | 172.24.1.12 |
| compute0.overcloud.example.com    | 172.24.1.2  |
+-----------------------------------+-------------+
```

```
[student@workstation ~(architect1-finance)]$ openstack network list -c ID -c Name
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
...output omitted...
| ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00 | provider-datacentre |
+--------------------------------------+---------------------+
```

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/

DESCRIBING NETWORK PROTOCOL TYPES

In this exercise, you will deploy two instances and view the components of the path from one instance to the other.

OUTCOMES

You should be able to:

- View the components of the controller node.
- View the components of the compute nodes.
- View the Geneve network packets.

BEFORE YOU BEGIN

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

Run the **`lab networking-protocols setup`** command. This script ensures that the OpenStack services are running and the environment is properly configured for this exercise.

```
[student@workstation ~]$ lab networking-protocols setup
```

1. On `workstation`, source the **`architect1-finance-rc`** identity environment file. A zone can only be specified when using an admin account. As the `architect1` user, create two instances. Use the `rhel7` image, the `default` flavor, and the `finance-network1` network. Create `finance-server3` on `compute0` and `finance-server4` on `compute1`. Take note of the IP addresses because they are used in future steps.

```
[student@workstation ~]$ source /home/student/architect1-finance-rc
[student@workstation ~(architect1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--nic net-id=finance-network1 \
--availability-zone nova:compute0.overcloud.example.com \
--wait finance-server3
+-------------------------------------+--------------------------------+
| Field                               | Value                          |
+-------------------------------------+--------------------------------+
| OS-DCF:diskConfig                   | MANUAL                         |
| OS-EXT-AZ:availability_zone         | nova                           |
| OS-EXT-SRV-ATTR:host                | compute0.overcloud.example.com |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute0.overcloud.example.com |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000007              |
| OS-EXT-STS:power_state              | Running                        |
| OS-EXT-STS:task_state               | None                           |
| OS-EXT-STS:vm_state                 | active                         |
```

```
| OS-SRV-USG:launched_at              | 2018-10-04T09:06:51.000000      |
| OS-SRV-USG:terminated_at            | None                            |
| accessIPv4                          |                                 |
| accessIPv6                          |                                 |
| addresses                           | finance-network1=192.168.1.11   |
| adminPass                           | PZgvW7cp287V                    |
| config_drive                        |                                 |
| created                             | 2018-10-04T09:05:44Z            |
| flavor                              | default (7ac0...0a9e)           |
| hostId                              | 7f59...4ca01                    |
| id                                  | bfa1...c531                     |
| image                               | rhel7 (dd3d...d02e)             |
| key_name                            | None                            |
| name                                | finance-server3                 |
| progress                            | 0                               |
| project_id                          | f8594af5ef6246888ed31df520fccf30|
| properties                          |                                 |
| security_groups                     | name='default'                  |
| status                              | ACTIVE                          |
| updated                             | 2018-10-04T09:06:51Z            |
| user_id                             | 8eb50062824b468b9b205c538d57f46f|
| volumes_attached                    |                                 |
+-------------------------------------+---------------------------------+
```

```
[student@workstation ~(architect1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--nic net-id=finance-network1 \
--availability-zone nova:compute1.overcloud.example.com \
--wait finance-server4
+-------------------------------------+---------------------------------+
| Field                               | Value                           |
+-------------------------------------+---------------------------------+
| OS-DCF:diskConfig                   | MANUAL                          |
| OS-EXT-AZ:availability_zone         | nova                            |
| OS-EXT-SRV-ATTR:host                | compute1.overcloud.example.com  |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute1.overcloud.example.com  |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000008               |
| OS-EXT-STS:power_state              | Running                         |
| OS-EXT-STS:task_state               | None                            |
| OS-EXT-STS:vm_state                 | active                          |
| OS-SRV-USG:launched_at              | 2018-10-04T09:12:11.000000      |
| OS-SRV-USG:terminated_at            | None                            |
| accessIPv4                          |                                 |
| accessIPv6                          |                                 |
| addresses                           | finance-network1=192.168.1.8    |
| adminPass                           | UBzit2gsUGKe                    |
| config_drive                        |                                 |
| created                             | 2018-10-04T09:11:43Z            |
| flavor                              | default (7ac0...0a9e)           |
| hostId                              | eeac...f6a7                     |
| id                                  | 3f0c...e004                     |
| image                               | rhel7 (dd3d...d02e)             |
| key_name                            | None                            |
| name                                | finance-server4                 |
```

```
| progress                            | 0                                |
| project_id                          | f8594af5ef6246888ed31df520fccf30|
| properties                          |                                  |
| security_groups                     | name='default'                   |
| status                              | ACTIVE                           |
| updated                             | 2018-10-04T09:12:12Z             |
| user_id                             | 8eb50062824b468b9b205c538d57f46f|
| volumes_attached                    |                                  |
+-------------------------------------+----------------------------------+
```

2. Use the **ovs-ofctl** and **ovs-vsctl** commands to view the components on the overcloud nodes.

   2.1. Open another terminal. From workstation, log in to controller0 as the heat-admin user. Use the **sudo -i** command to gain root privileges.

```
[student@workstation ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

   2.2. Use the **ovs-vsctl show** command to list the configuration of the bridges on the controller0 node. Notice the configuration of br-ex.

```
[root@controller0 ~]# ovs-vsctl show
...output omitted...
Bridge br-ex
    fail_mode: standalone
    Port "eth2"
         Interface "eth2"
    Port br-ex
         Interface br-ex
         type: internal
    Port "patch-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00-to-br-int"
         Interface "patch-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00-to-br-int"
         type: patch
         options: {peer="patch-br-int-to-provnet-ef9b4ad2-36c0-4ed4-8547-
d7846f7d0f00"}
...output omitted...
```

   On workstation, use the **openstack network list** command to list the OpenStack networks. Notice that the ID of provider-datacentre matches one of the ports on br-ex.

```
[student@workstation ~(architect1-finance)]$ openstack network list -c ID -c Name
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 1c458a95-a00f-411e-8a73-d88ba18dd0a0 | finance-network1    |
| 772fe396-a9dc-4243-80bc-682664829e00 | lb-mgmt-net         |
| b5522075-961a-40f3-b7b4-ce5236a3f73c | production-network1 |
| e8d43fea-ad78-4165-913f-ad74288d0e51 | demo-network1       |
| ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00 | provider-datacentre |
```

```
+-------------------------------------+--------------------+
```

2.3. On `controller0`, use the **`ovs-vsctl show`** command to list the configuration of the bridges on the `controller0` node. Notice the configuration of `br-int`.

```
[root@controller0 ~]# ovs-vsctl show
...output omitted...
Bridge br-int
    fail_mode: secure
    Port "ovn-840e32-0"
         Interface "ovn-840e32-0"
         type: geneve
         options: {csum="true", key=flow, remote_ip="172.24.2.2"}
    Port "ovn-14c844-0"
         Interface "ovn-14c844-0"
         type: geneve
         options: {csum="true", key=flow, remote_ip="172.24.2.12"}
    Port "o-hm0"
         Interface "o-hm0"
         type: internal
    Port "ovn-263d95-0"
         Interface "ovn-263d95-0"
         type: geneve
         options: {csum="true", key=flow, remote_ip="172.24.2.6"}
    Port br-int
         Interface br-int
         type: internal
    Port "patch-br-int-to-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00"
         Interface "patch-br-int-to-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00"
         type: patch
         options: {peer="patch-provnet-ef9b4ad2-36c0-4ed4-8547-d7846f7d0f00-to-br-
int"}
  ovs_version: "2.9.0"
...output omitted...
```

On `workstation`, use the **`openstack hypervisor list`** command to list the compute nodes. Notice that interface `ovn-840e32-0` corresponds to `compute0`, `ovn-14c844-0` corresponds to `compute1`, and `ovn-263d95-0` corresponds to `computehci0`. Also note that all of those interfaces are of type Geneve. There is also a port for the interface of the `provider-datacentre` network.

```
[student@workstation ~(architect1-finance)]$ openstack hypervisor list \
-c "Hypervisor Hostname" -c "Host IP"
+-----------------------------------+-------------+
| Hypervisor Hostname               | Host IP     |
+-----------------------------------+-------------+
| computehci0.overcloud.example.com | 172.24.1.6  |
| compute1.overcloud.example.com    | 172.24.1.12 |
| compute0.overcloud.example.com    | 172.24.1.2  |
+-----------------------------------+-------------+
```

2.4. On `controller0`, use the **`ovs-ofctl show`** command to list the interfaces on `br-trunk`. Notice the five VLAN interfaces.

```
>[root@controller0 ~]# ovs-ofctl show br-trunk
```

```
OFPT_FEATURES_REPLY (xid=0x2): dpid:0000525400010001
n_tables:254, n_buffers:0
capabilities: FLOW_STATS TABLE_STATS PORT_STATS QUEUE_STATS ARP_MATCH_IP
actions: output enqueue set_vlan_vid set_vlan_pcp strip_vlan mod_dl_src mod_dl_dst
 mod_nw_src mod_nw_dst mod_nw_tos mod_tp_src mod_tp_dst
 1(eth1): addr:52:54:00:01:00:01
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 2(vlan10): addr:7a:cf:bb:f5:83:cc
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 3(vlan20): addr:22:0c:51:bd:2e:bd
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 4(vlan30): addr:92:b5:04:bc:23:82
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 5(vlan40): addr:52:ff:78:53:14:ea
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 6(vlan50): addr:7e:6b:6e:17:74:59
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 LOCAL(br-trunk): addr:52:54:00:01:00:01
    config:     0
    state:      0
    speed: 0 Mbps now, 0 Mbps max
 OFPT_GET_CONFIG_REPLY (xid=0x4): frags=normal miss_send_len=0
```

2.5. Open another terminal window. From `workstation`, log in to `compute0` as the `heat-admin` user. Use the **`ip link`** command to list the interfaces. Notice the VLAN interfaces. VLAN20 is the tenant VLAN.

```
[student@workstation ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ ip link
...output omitted...
14: vlan10: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
 mode DEFAULT group default qlen 1000
    link/ether 1a:e5:09:ff:90:a3 brd ff:ff:ff:ff:ff:ff
15: vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
 mode DEFAULT group default qlen 1000
    link/ether 8e:43:4c:8b:d8:cf brd ff:ff:ff:ff:ff:ff
16: vlan30: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
 mode DEFAULT group default qlen 1000
    link/ether ca:ca:f9:af:53:0e brd ff:ff:ff:ff:ff:ff
17: vlan50: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UNKNOWN
 mode DEFAULT group default qlen 1000
    link/ether ca:85:82:a1:f3:45 brd ff:ff:ff:ff:ff:ff
```

3. Capture packets on interface `VLAN20` to show traffic between `finance-server3` and `finance-server4`.

3.1. On `compute0`, use the **`sudo -i`** command to gain root privileges. Use the **`tcpdump`** command to capture packets on the `VLAN20` tenant network.

```
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]# tcpdump -ten -i vlan20 | grep ICMP
tcpdump: listening on eth1, link-type EN10MB (Ethernet), capture size 262144 bytes
```

3.2. On `workstation`, use the **`openstack console url show`** command to obtain the console URL for `finance-server3`. Open the URL in Firefox, then log in to `finance-server3` using `root` as the username and `redhat` as the password. Ping `finance-server4`.

```
[student@workstation ~(architect1-finance)]$ openstack console url show \
finance-server3 --max-width=80
+-------+-----------------------------------------------------------------+
| Field | Value                                                           |
+-------+-----------------------------------------------------------------+
| type  | novnc                                                           |
| url   | http://172.25.250.50:6080/vnc_auto.html?token=201c4763-7f10-4c55 |
|       | -a3fb-a4fd045a5f32                                              |
+-------+-----------------------------------------------------------------+
[student@workstation ~(architect1-finance)]$ firefox \
http://172.25.250.50:6080/vnc_auto.html?token=201c4763-7f10-4c55-a3fb-a4fd045a5f32
```

```
[root@finance-server3 ~]$ ping -c3 192.168.1.8
64 bytes from 192.168.1.8: icmp_seq=1 ttl=64 time=0.450 ms
64 bytes from 192.168.1.8: icmp_seq=1 ttl=64 time=0.350 ms
64 bytes from 192.168.1.8: icmp_seq=1 ttl=64 time=0.430 ms
...output omitted...
```

3.3. On `compute0`, analyze the output of the **`tcpdump`** command. Look for the *ICMP echo request* and *ICMP echo reply* packets between `finance-server3` and `finance-server4`. Notice that the packets are sent between `compute0` and `compute1`. The tunnel type is Geneve, and the **`vni`** parameter holds the ID for the packet.

```
8e:43:4c:8b:d8:cf > 76:3a:02:69:82:1e, ethertype IPv4 (0x0800), length
 156: 172.24.2.2.35614 > 172.24.2.12.6081: Geneve, Flags [C], vni 0x3, proto TEB
 (0x6558), options [8 bytes]: fa:16:3e:b5:fd:da > fa:16:3e:ec:5f:e0, ethertype
 IPv4 (0x0800), length 98: 192.168.1.11 > 192.168.1.8: ICMP echo request, id 1211,
 seq 1, length 64
         76:3a:02:69:82:1e > 8e:43:4c:8b:d8:cf, ethertype IPv4 (0x0800), length
 156: 172.24.2.12.23252 > 172.24.2.2.6081: Geneve, Flags [C], vni 0x3, proto TEB
 (0x6558), options [8 bytes]: fa:16:3e:ec:5f:e0 > fa:16:3e:b5:fd:da, ethertype
 IPv4 (0x0800), length 98: 192.168.1.8 > 192.168.1.11: ICMP echo reply, id 1211,
 seq 1, length 64
```

4. Open another terminal. From `workstation`, log in to `compute1` as the `heat-admin` user. Use the **`sudo -i`** command to gain root privileges.

```
[student@workstation ~]$ ssh heat-admin@compute1
```

```
[heat-admin@compute1 ~]$ sudo -i
[root@compute1 ~]#
```

5. On `compute0`, use the **`ovs-vsctl list open`** command to find the IP address of the Southbound and Northbound OVN databases. The IP address is in the **`ovn-remote`** parameter. Set the environment variables for the OVN databases on both `compute0` and `compute1`. The port for the Northbound database is 6641.

```
[root@compute0 ~]# ovs-vsctl list open
...output omitted...
external_ids        : {hostname="compute0.overcloud.example.com", ovn-bridge=br-
int, ovn-bridge-mappings="datacentre:br-ex,provider_vlan:br-eth3", ovn-encap-
ip="172.24.2.2", ovn-encap-type=geneve, ovn-remote="tcp:172.24.1.50:6642",
 rundir="/var/run/openvswitch", system-id="840e32d5-21c3-4760-b6e8-8bfd063b50a1"}
...output omitted...
```

```
[root@compute0 ~]# export OVN_SB_DB=tcp:172.24.1.50:6642
[root@compute0 ~]# export OVN_NB_DB=tcp:172.24.1.50:6641
```

6. On `workstation`, use the **`openstack port list`** command to find the port ID for `finance-server3` and `finance-server4`.

```
[student@workstation ~(architect1-finance)]$ openstack port list \
-c ID -c "Fixed IP Addresses"
+-------------+-------------------------------------------------------+
| ID          | Fixed IP Addresses                                    |
+-------------+-------------------------------------------------------+
| f13c...9107 | ip_address='192.168.1.11', subnet_id='1479...6a78'    |
...output omitted...
| 9b27...78d7 | ip_address='192.168.1.8', subnet_id='1479...6a78'     |
...output omitted...
```

7. Use the **`ovn-sbctl lflow-list`** command to list the OpenFlow flows for the Southbound database.

   7.1. On `compute0`, use the **`ovn-sbctl lflow-list`** command to list the OpenFlow flows. Redirect the output to a file called **`flow.txt`**.

```
[root@compute0 ~]# ovn-sbctl lflow-list > flow.txt
```

   7.2. Search the file for the ID of `finance-server3`. Search the flows to find the tables that allow traffic from `finance-server3` and `finance-server4`.

```
[root@compute0 ~]# view flow.txt
...output omitted...
  table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport
 == "f13cebbd-32ca-4fc8-9a65-4293f4fe9107" && eth.src ==
 {fa:16:3e:bc:bb:d9}), action=(next;)
...output omitted...
  table=13(ls_in_dhcp_response), priority=100  , match=(inport ==
 "f13cebbd-32ca-4fc8-9a65-4293f4fe9107" && eth.src == fa:16:3e:bc:bb:d9 && ip4 &&
 udp.src == 68 && udp.dst == 67 && reg0[3]), action=(eth.dst = eth.src; eth.src =
```

```
 fa:16:3e:f8:63:f0; ip4.dst = 192.168.1.11; ip4.src = 192.168.1.1; udp.src = 67;
 udp.dst = 68; outport = inport; flags.loopback = 1; output;)
  table=13(ls_in_dhcp_response), priority=0    , match=(1), action=(next;)
  table=14(ls_in_dns_lookup   ), priority=0    , match=(1), action=(next;)
  table=15(ls_in_dns_response ), priority=0    , match=(1), action=(next;)
  table=16(ls_in_l2_lkup      ), priority=100  , match=(eth.mcast),
 action=(outport = "_MC_flood"; output;)
  table=16(ls_in_l2_lkup      ), priority=50   , match=(eth.dst ==
 fa:16:3e:18:e6:f6), action=(outport = "9b272b18-1b7c-47a9-9d9f-d431d31278d7";
 output;)
...output omitted...
```

7.3. Repeat the above steps on `compute1`. Look for the OpenFlow flows for `finance-server4`.

```
[root@compute1 ~]# ovn-sbctl lflow-list > flow.txt
[root@compute1 ~]# view flow.txt
...output omitted...
  table=0 (ls_in_port_sec_l2  ), priority=50   , match=(inport
 == "9b272b18-1b7c-47a9-9d9f-d431d31278d7" && eth.src ==
 {fa:16:3e:18:e6:f6}), action=(next;)
...output omitted...
  table=13(ls_in_dhcp_response), priority=100  , match=(inport ==
 "9b272b18-1b7c-47a9-9d9f-d431d31278d7" && eth.src == fa:16:3e:18:e6:f6 && ip4 &&
 udp.src == 68 && udp.dst == 67 && reg0[3]), action=(eth.dst = eth.src; eth.src
 = fa:16:3e:f8:63:f0; ip4.dst = 192.168.1.8; ip4.src = 192.168.1.1; udp.src = 67;
 udp.dst = 68; outport = inport; flags.loopback = 1; output;)
  table=13(ls_in_dhcp_response), priority=100  , match=(inport ==
 "f13cebbd-32ca-4fc8-9a65-4293f4fe9107" && eth.src == fa:16:3e:bc:bb:d9 && ip4 &&
 udp.src == 68 && udp.dst == 67 && reg0[3]), action=(eth.dst = eth.src; eth.src =
 fa:16:3e:f8:63:f0; ip4.dst = 192.168.1.11; ip4.src = 192.168.1.1; udp.src = 67;
 udp.dst = 68; outport = inport; flags.loopback = 1; output;)
  table=13(ls_in_dhcp_response), priority=0    , match=(1), action=(next;)
  table=14(ls_in_dns_lookup   ), priority=0    , match=(1), action=(next;)
  table=15(ls_in_dns_response ), priority=0    , match=(1), action=(next;)
  table=16(ls_in_l2_lkup      ), priority=100  , match=(eth.mcast),
 action=(outport = "_MC_flood"; output;)
  table=16(ls_in_l2_lkup      ), priority=50   , match=(eth.dst ==
 fa:16:3e:18:e6:f6), action=(outport = "9b272b18-1b7c-47a9-9d9f-d431d31278d7";
 output;)
...output omitted...
```

8. Using the same file on each compute node, find the security group flows for DHCP, SSH, and ICMP.

```
[root@compute0 ~]# view flow.txt
...output omitted...
  table=6 (ls_in_acl          ), priority=2002 , match=(!ct.new
 && ct.est && !ct.rpl && ct_label.blocked == 0 && (inport ==
 "f13cebbd-32ca-4fc8-9a65-4293f4fe9107" && ip4 && ip4.dst == {255.255.255.255,
 192.168.1.0/24} && udp && udp.src == 68 && udp.dst == 67)), action=(next;)
...output omitted...
```

```
  table=4 (ls_out_acl         ), priority=2002 , match=(!ct.new  ct.est && !ct.rpl
 && ct_label.blocked == 0 && (outport == "f13cebbd-32ca-4fc8-9a65- 4293f4fe9107"
 && ip4 && ip4.src == 0.0.0.0/0 && icmp4)), action=(next;)
  table=4 (ls_out_acl         ), priority=2002 , match=(!ct.new
 && ct.est && !ct.rpl && ct_label.blocked == 0 && (outport ==
 "f13cebbd-32ca-4fc8-9a65-4293f4fe9107" && ip4 && ip4.src == 0.0.0.0/0 && tcp &&
 tcp.dst == 22)), action=(next;)
...output omitted...
```

```
table=6 (ls_in_acl          ), priority=2002 , match=(!ct.new && ct.est && !ct.rpl
 && ct_label.blocked == 0 && (inport == "9b272b18-1b7c-47a9-9d9f-d431d31278d7" &&
 ip4 && ip4.dst == {255.255.255.255, 192.168.1.0/24} && udp && udp.src == 68 &&
 udp.dst == 67)), action=(next;)
...output omitted...
  table=4 (ls_out_acl         ), priority=2002 , match=(!ct.new && ct.est
 && !ct.rpl && ct_label.blocked == 0 && (outport == "9b272b18-1b7c-47a9-9d9f-
d431d31278d7" && ip4 && ip4.src == 0.0.0.0/0 && icmp4)), action=(next;)
  table=4 (ls_out_acl         ), priority=2002 , match=(!ct.new && ct.est
 && !ct.rpl && ct_label.blocked == 0 && (outport == "9b272b18-1b7c-47a9-9d9f-
d431d31278d7" && ip4 && ip4.src == 0.0.0.0/0  tcp && tcp.dst == 22)),
 action=(next;)
...output omitted...
```

9. On `compute0`, use the **`ovn-nbctl show`** command to list the contents of the Northbound database. Find the `lrp` (logical router port) port for `finance-network1`. On `compute1`, in the **`flow.txt`** file, find the OpenFlow flows for the port for `finance-network1`.

```
[root@compute0 ~]# ovn-nbctl show
...output omitted...
switch 3bc3a1b0-2a6f-4b91-853e-b2d143871066 (neutron-1c458a95-a00f-411e-8a73-
d88ba18dd0a0) (aka finance-network1)
   port 9b272b18-1b7c-47a9-9d9f-d431d31278d7
       addresses: ["fa:16:3e:18:e6:f6 192.168.1.8"]
   port f13cebbd-32ca-4fc8-9a65-4293f4fe9107
       addresses: ["fa:16:3e:bc:bb:d9 192.168.1.11"]
   port 252e3397-e320-4b99-ae82-ee3a2e91d565
        type: router
        router-port: lrp-252e3397-e320-4b99-ae82-ee3a2e91d565
   port 3d58e426-a8d7-4ea3-a974-0abd48b5b1d3
        type: localport
        addresses: ["fa:16:3e:54:e9:70 192.168.1.2"]
...output omitted...
  table=8 (lr_in_arp_resolve  ), priority=100  , match=(outport == "lrp-252e3397-
e320-4b99-ae82-ee3a2e91d565" && reg0 == 192.168.1.8), action=(eth.dst =
 fa:16:3e:18:e6:f6; next;)
...output omitted...
  table=8 (lr_in_arp_resolve  ), priority=100  , match=(outport == "lrp-252e3397-
e320-4b99-ae82-ee3a2e91d565" && reg0 == 192.168.1.11), action=(eth.dst =
 fa:16:3e:bc:bb:d9; next;)
...output omitted...
```

10. Close all of the terminal windows opened during this exercise.

Cleanup

On *workstation*, run the **`lab networking-protocols cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab networking-protocols cleanup
```

This concludes the guided exercise.

INTRODUCING OPEN VIRTUAL NETWORKING (OVN)

OBJECTIVES

After completing this section, students should be able to describe the OVN architecture and how it integrates with OVS and OpenStack neutron.

INTRODUCTION TO MODULAR LAYER 2 (ML2)

The *Modular Layer 2 (ML2)* plug-in is a framework that enables the use of various technologies. For example, administrators can interact with Open vSwitch, a technology that provides virtual switching, or Cisco equipment, using the various plug-ins available for OpenStack Networking.

ML2 Drivers and Networks Types

The ML2 framework simplifies support for layer 2 (L2) networking technologies. The framework creates an abstraction layer which separates the management of network types from the mechanisms used to access these networks. The framework allows multiple mechanism drivers to access the same networks simultaneously. The implementation of ML2 allows companies and manufacturers to develop their own plug-ins.

The ML2 plug-in currently includes drivers for the following network types:

Flat

A flat network is one that does not support segmentation. A traditional layer 2 Ethernet network is a flat network. Servers connected to flat networks listen to the broadcast traffic and can contact each other. In OpenStack terminology, flat networks connect instances to existing layer 2 networks, or *provider* networks.

VLAN

A VLAN network uses VLANs for segmentation. A VLAN *identifier* (ID) is assigned to each network. The ID is selected from the range defined in the OpenStack Networking configuration. Administrators must configure the network switches to trunk the corresponding VLANs.

GRE and VXLAN

These networks are similar to VLAN networks. GRE and VXLAN are overlay networks which encapsulate network traffic. Both networks receive a unique Virtual Network Identifier (VNI). However, unlike VLANs, overlay networks do not require any synchronization between the OpenStack environment and layer 2 switches.

Geneve

Geneve (Generic Network Virtualization Encapsulation) addresses the limitations of the other network types. Geneve defines an encapsulation format only. It does not include any specification for the control plane, unlike all other network types. All encapsulated packets are forwarded using standard network equipment. Like VLAN, GRE and VXLAN networks the Geneve network receives a unique tunnel identifier (VIN). The tunnel endpoints communicate only with each other. Geneve is the default tunneling program for *OVN*.

INTRODUCTION TO OPEN VIRTUAL NETWORKING (OVN)

OVN is an open source project, launched by the *Open vSwitch* team. The idea was to create a vendor neutral protocol for the virtualization of virtual switching. It provides Layer2 and Layer3 networking, where other *Software Defined Network (SDN)* solutions commonly provide only Layer2

or Layer3. OVN allows for the implementation of security groups, and includes a DHCP service, L3 routing, and NAT. In Red Hat OpenStack, OVN exclusively uses the Geneve tunnel overlay network.

OVN is the default SDN solution for Red Hat OpenStack. It replaces the OVS ML2 driver and the neutron agent with the OVN ML2 driver. The transition to OVN is natural and seamless because it compliments the existing Open vSwitch technology already implemented in OpenStack. Scalability is improved compared to other SDN solutions as it does not use the neutron agents. Instead it uses the `ovn-controller` and OVS flows to implement all functionality. It connects to VMs and containers without provisioning physical network resources, which helps to improve performance.

OVN Architecture

The OpenStack networking configuration is translated into an OVN logical networking configuration using the *OVN ML2 Plug-in*. The plug-in runs on the controller nodes.

The *OVN Northbound (NB) database* stores the logical OVN networking configuration, which it gets from the OVN ML2 plug-in. The plug-in runs on the controller nodes and listens on TCP port 6641.

The *OVN Northbound service* converts the logical network configuration from the OVN NB database to the logical path flows. The `ovn-northd` populates the *OVN Southbound database* with the logical path flows. The service runs on the controller nodes.

The *OVN Southbound (SB) database* listens on port 6642. The `ovn-controller` connects to the Southbound database to control and monitor network traffic. This service runs on all compute and gateway nodes where **OS::Tripleo::Services::OVNController** is defined.

The *OVN metadata agent* spawns the *HAProxy* instances. These instances manage the OVS interfaces, network namespaces and HAProxy processes. It runs on all compute and gateway nodes where **OS::TripleO::Services::OVNMetadataAgent** is defined.

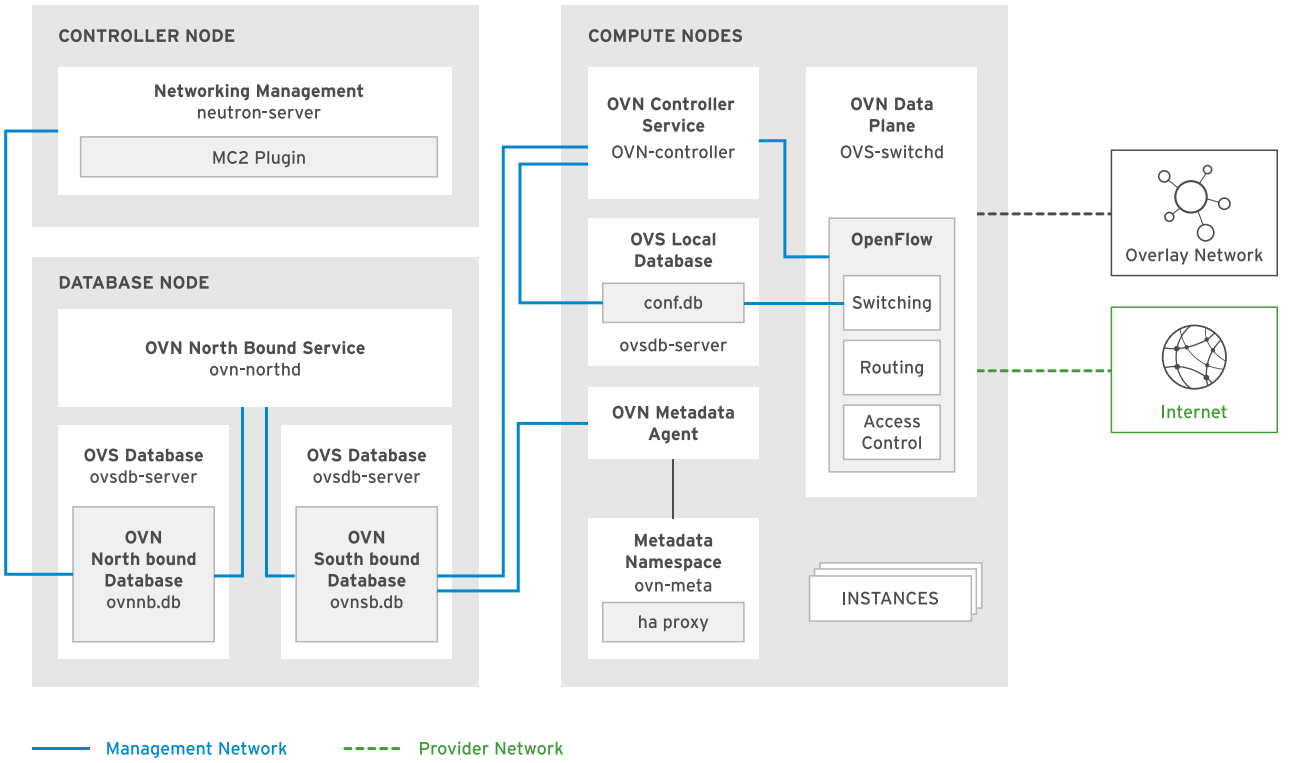


Figure 6.4: OVN architecture

OVN Database

The OVN database is installed in a central location. The database can be installed on a physical node, a virtual node, or on a cluster. The choice of location depends on various factors, including the size of the cloud infrastructure, the geographic dispersion of the cloud, the volume of traffic, and the performance required. The hypervisors must run Open vSwitch for OVN to work.

There are two parts to the OVN database: the *Northbound* Database and the *Southbound* Database. The Northbound Database receives information about the logical network configuration from the neutron plug-in. It has two clients, the neutron plug-in and `ovn-northd`. The `ovn-northd` client connects to the OVN Northbound Database and the OVN Southbound Database. It translates the logical network configuration into logical data path flows and stores them in the OVN Southbound Database.

The OVN Southbound database is the center of the entire system. It also has two clients, the `ovn-northd` and the `ovn-controller` services. Each hypervisor has its own `ovn-controller`. The database contains three types of data:

- **Physical Network** tables specifying how to reach the overcloud nodes.
- **Logical Network** tables specifying the logical data path flows.
- **Binding** tables linking the location of logical network components to the physical network.

The Physical Network tables and Binding tables are populated by the hypervisors. The Logical Network tables are populated by `ovn-northd`.

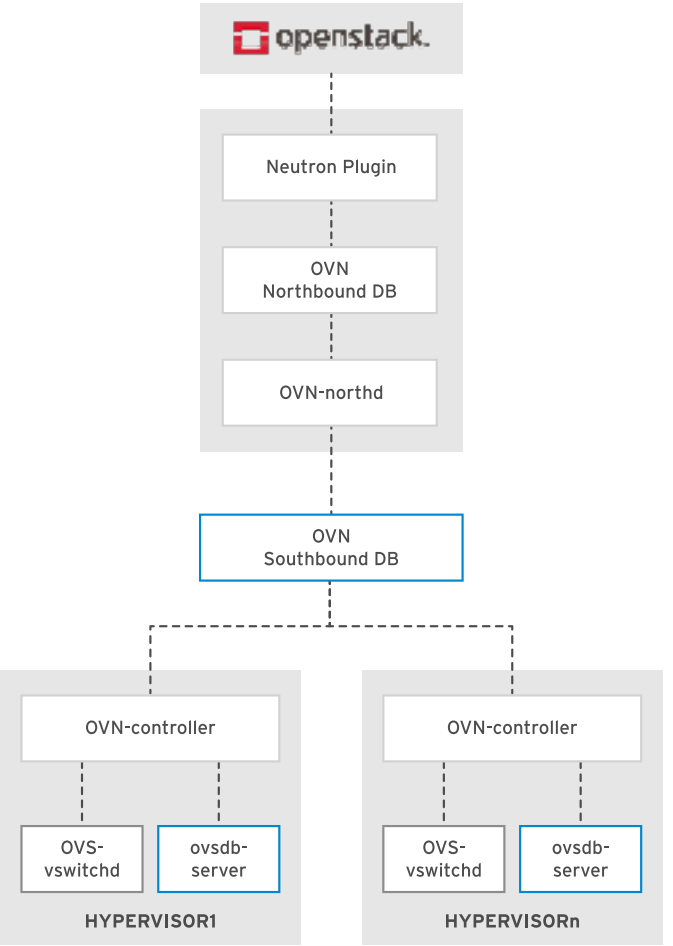


Figure 6.5: OVN control plane architecture

OVN and OpenFlow

In Red Hat OpenStack, OVN is managed with the *OpenFlow* protocol. The OpenFlow protocol is used to program the Open vSwitch pipeline, and so defines how network traffic should be handled. A series of flow tables exist, where each flow has a priority, a match, and a set of actions. The flow with the highest priority will be executed first. OpenFlow is capable of dynamically rewriting flow tables, allowing it to add and remove network functions as required.

OVN Logical Flows

Manually programming OpenFlow pipelines, or flows, is a tedious task. Creating them by hand, in an OpenStack environment, would be extremely complicated, time-consuming, and difficult to manage on a day-to-day basis. An SDN controller such as OVN, creates flows automatically across all switches and network components. OVN is also responsible for implementing security policies.

Logical flows are very similar to OpenFlow. They have priorities, a match, and actions. Logical flows also describe the detailed behavior of an entire network. OVN creates the entire network in logical flows which are distributed to the `ovn-controller` on each hypervisor. Each `ovn-controller` translates the logical flows into OpenFlow flows, detailing how to reach the other hypervisors.

OVN defines logical switches and ports, where both an ingress and an egress pipeline are created. A packet entering the network goes through the ingress pipeline on the hypervisor where the packet originated. Assuming the destination is on the same hypervisor the egress pipeline is executed. If the destination is on another hypervisor the packet is sent over a Geneve tunnel and the egress pipeline is executed on the remote host.

Use the **`ovs-vsctl list open`** command to find the IP address of the **`ovn-remote`** parameter. The OVN_SB_DC environment variable must be set to use the **`ovn`** commands. Set the environment variable, then use the **`ovn-sbctl lflow-list`** command to view the logical flows.

```
[root@controller0 ~]# ovs-vsctl list open
_uuid               : e6cff4b4-5e7e-4a06-9d34-7060543b85b5
bridges             : [4d55bad8-8dbb-45a6-a059-291f5cfd4722, 57c81206-96be-48eb-
abac-22b71be9f07e, ac93ee97-e01f-4cde-ab1e-f8a6d10b0558]
cur_cfg             : 46
datapath_types      : [netdev, system]
db_version          : "7.15.1"
external_ids        : {hostname="controller0.overcloud.example.com", ovn-
bridge=br-int, ovn-bridge-mappings="datacentre:br-ex", ovn-cms-options=enable-
chassis-as-gw, ovn-encap-ip="172.24.2.1", ovn-encap-type=geneve, ovn-
remote="tcp:172.24.1.50:6642", rundir="/var/run/openvswitch", system-
id="ee00e7d6-3fb8-46cd-b2d7-2d3d3b92eb5b"}
iface_types         : [geneve, gre, internal, lisp, patch, stt, system, tap,
 vxlan]
manager_options     : []
next_cfg            : 46
other_config        : {}
ovs_version         : "2.9.0"
ssl                 : []
statistics          : {}
system_type         : rhel
system_version      : "7.5"
[root@controller0 ~]# export OVN_SB_DB=tcp:172.24.1.50:6642
[root@controller0 ~]# ovn-sbctl lflow-list
...output omitted...
  table=11(ls_in_arp_rsp      ), priority=50   , match=(nd_ns && ip6.dst
 == {fe80::f816:3eff:fe97:4c1b, ff02::1:ff97:4c1b} && nd.target ==
 fe80::f816:3eff:fe97:4c1b), action=(nd_na { eth.src = fa:16:3e:97:4c:1b; ip6.src
```

```
  = fe80::f816:3eff:fe97:4c1b; nd.target = fe80::f816:3eff:fe97:4c1b; nd.tll =
  fa:16:3e:97:4c:1b; outport = inport; flags.loopback = 1; output; };)
     table=11(ls_in_arp_rsp      ), priority=0    , match=(1), action=(next;)
       table=12(ls_in_dhcp_options ), priority=100  , match=(inport ==
  "79bb40e8-343a-4d7b-810a-44412ca470b6" && eth.src == fa:16:3e:8b:61:a5 && ip4.src
  == 0.0.0.0 && ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst == 67),
  action=(reg0[3] = put_dhcp_opts(offerip = 192.168.1.6, classless_static_route
  = {169.254.169.254/32,192.168.1.2, 0.0.0.0/0,192.168.1.1}, dns_server =
  {172.25.250.254}, lease_time = 43200, mtu = 1442, netmask = 255.255.255.0, router
  = 192.168.1.1, server_id = 192.168.1.1); next;)
         table=12(ls_in_dhcp_options ), priority=100  , match=(inport ==
  "79bb40e8-343a-4d7b-810a-44412ca470b6" && eth.src == fa:16:3e:8b:61:a5 &&
  ip4.src == 192.168.1.6 && ip4.dst == {192.168.1.1, 255.255.255.255} && udp.src
  == 68 && udp.dst == 67), action=(reg0[3] = put_dhcp_opts(offerip = 192.168.1.6,
  classless_static_route = {169.254.169.254/32,192.168.1.2, 0.0.0.0/0,192.168.1.1},
  dns_server = {172.25.250.254}, lease_time = 43200, mtu = 1442, netmask =
  255.255.255.0, router = 192.168.1.1, server_id = 192.168.1.1); next;)
 ...output omitted...
```

COMPARING ML2/OVS TO ML2/OVN

There are many differences between ML2 using OVS and ML2 using OVN. These differences affect how administrators carry out various tasks. The changes are especially important during troubleshooting. OpenFlow rules manage outgoing and incoming traffic, security policy implementation, DHCP, and NAT. This means that the number of rules have increased greatly in comparison to OVS.

Key Differences

COMPONENT	ML2 WITH OVS	ML2 WITH OVN
agent/server communication	Uses the RabbitMQ messaging backend.	The ovsdb protocol is used.
L3HA dataplane	Implemented by creating a qrouter namespace.	The ovn-controller configures OpenFlow rules.
DVR API	The "distributed" flag is modifiable by an admin.	All traffic is distributed.
DVR data plane	Composed of namespaces, veth pairs, and iptables rules.	Composed of OpenFlow rules on the compute nodes.
East/West traffic	Traffic is routed through network nodes when DVR is switched off.	Traffic is distributed in all cases.
metadata service	Supported by DHCP namespaces on the controller nodes.	Runs on all compute nodes within the ovnmeta-xxx namespace.
DHCP Service	Provided by dhcp-xxx namespaces, with a dnsmasq process running inside each namespace.	Implemented using OpenFlow rules interpreted by the ovn-controller, and distributed across all compute nodes.

OVN GATEWAY ROUTER

The OVN Gateway links the overlay network, managed by `ovn-northd`, to the physical network. There are two ways to link the overlay network to the physical network. Using layer 2 to bridge an OVN logical switch into a VLAN, or at layer 3 by providing a connection between an OVN router and the physical network.

OVN has three types of NAT:

- DNAT, which changes the destination address of a packet
- SNAT, which changes the source address of a packet
- **SNAT-DNAT**, which creates a static NAT. An external IP is mapped to an internal IP, or vice versa.

OVN OpenFlow flows manage all types of NAT. Use the **`ovn-sbctl lflow-list`** command to see the relevant flows. There are SNAT and DNAT flows for each virtual machine created.

```
table=1 (lr_out_snat        ), priority=33   , match=(ip && ip4.src == 192.168.1.6
 && outport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"), action=(eth.src =
 fa:16:3e:f5:58:23; ct_snat(172.25.250.106);)
  table=1 (lr_out_snat        ), priority=25   , match=(ip && ip4.src ==
 192.168.1.0/24 && outport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"
 && is_chassis_resident("cr-lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc")),
 action=(ct_snat(172.25.250.102);)
  table=1 (lr_out_snat        ), priority=0    , match=(1), action=(next;)
  table=3 (lr_in_unsnat       ), priority=100  , match=(ip && ip4.dst ==
 172.25.250.102 && inport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"
 && is_chassis_resident("cr-lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc")),
 action=(ct_snat;)
  table=3 (lr_in_unsnat       ), priority=50   , match=(ip && ip4.dst ==
 172.25.250.102), action=(reg9[0] = 1; next;)
  table=3 (lr_in_unsnat       ), priority=50   , match=(ip && ip4.dst ==
 172.25.250.106), action=(reg9[0] = 1; next;)
  table=3 (lr_in_unsnat       ), priority=50   , match=(ip && ip4.dst ==
 172.25.250.111), action=(reg9[0] = 1; next;)
  table=3 (lr_in_unsnat       ), priority=0    , match=(1), action=(next;)
  table=0 (lr_out_undnat      ), priority=100  , match=(ip && ip4.src ==
 192.168.1.6 && outport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"),
 action=(eth.src = fa:16:3e:f5:58:23; ct_dnat;)
  table=0 (lr_out_undnat      ), priority=0    , match=(1), action=(next;)
  table=4 (lr_in_dnat         ), priority=100  , match=(ip && ip4.dst ==
 172.25.250.106 && inport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"),
 action=(ct_dnat(192.168.1.6);)
  table=4 (lr_in_dnat         ), priority=50   , match=(ip && ip4.dst ==
 172.25.250.106), action=(reg9[0] = 1; next;)
  table=4 (lr_in_dnat         ), priority=50   , match=(ip && ip4.dst ==
 172.25.250.111), action=(reg9[0] = 1; next;)
  table=4 (lr_in_dnat         ), priority=0    , match=(1), action=(next;)
```

Notice that the inport and outport parameters correspond to one of the router ports for the `provider-datacentre` network. Use the **`ovn-nbctl lr-nat-list neutron-`*`54a04000-17ea-44df-8c79-d18164e6a1f2`*** command to list the NATs on a specific router.

```
switch ce4e7293-9bed-4504-9272-1fc698393048 (neutron-
f8366a5d-9cf9-4d38-9760-9cf9df4eceaa) (aka provider-datacentre)
   port provnet-f8366a5d-9cf9-4d38-9760-9cf9df4eceaa
```

```
        type: localnet
        addresses: ["unknown"]
    port 059d906d-fb1d-4120-8c74-5cd9de12f832
        type: localport
        addresses: ["fa:16:3e:e6:57:f1"]
    port 8905aec6-0db1-475f-8e38-0dfd4b4fe4fc
        type: router
        router-port: lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc
    port 8e0a4263-3b98-49cc-81a9-499fec769040
        type: router
        router-port: lrp-8e0a4263-3b98-49cc-81a9-499fec769040
```

Use the **`ovn-nbctl lr-list`** command to list the routers.

```
TYPE           EXTERNAL_IP    LOGICAL_IP     EXTERNAL_MAC       LOGICAL_PORT
dnat_and_snat 172.25.250.106 192.168.1.6    fa:16:3e:f5:58:23  79bb...470b6
snat           172.25.250.102 192.168.1.0/24
```

INTRODUCTION TO SECURITY GROUPS

Security groups and security rules filter the type and direction of network traffic sent to, and received from, an OpenStack Networking port. Support for security groups is performed by OVN and OpenStack neutron.

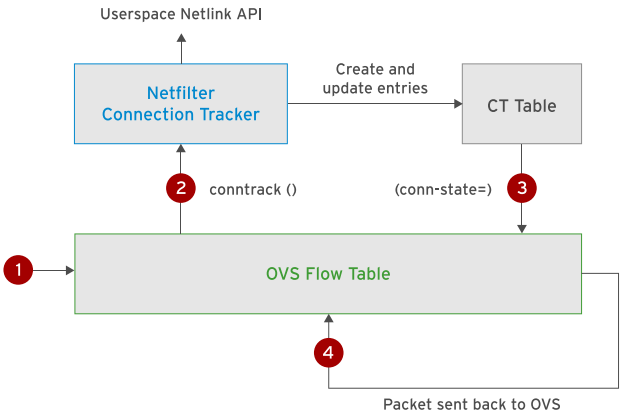
> **NOTE**
> By default, the group drops all inbound traffic and allows all outbound traffic.

Implementing Security Groups

In previous versions of OpenStack, support for security groups was implemented by OVS and ML2 using `iptables` rules. The iptables rules could only be applied to a Linux bridge so the VMs were connected to a Linux bridge using a tap device, then the Linux bridge was connected to the OVS bridge using a veth pair. These extra layers resulted in a more complex architecture.

To remove these extra layers, a firewall was implemented in OVS. However, OVS was only designed to support stateless matches, and firewalls require the *state* of a packet. To implement a firewall in OVS, support for stateful matches is necessary. The `conntrack` module from Linux is integrated into OVS to provide stateful matches. This module, also used by `iptables`, keeps track of all connections. The module does not enforce policies, but rather is concerned with whether or not a connection is *new*, *established*, or *related*. It adds the state of the connection to the metadata of the packet, which allows OVS to perform stateless matching on stateful traffic.

As with previous versions of OVS, traffic comes into the OVS Flow Table. However, to enforce the security policy OVS now requires further information, specifically the *state* of the packet in relation to existing connections. New OpenFlow extensions send an action to the conntrack module, where the state of the connection is added to the packet and the *Connection Tracker* table. The packet is then sent back to the OVS Flow Table with the connection status included, and the security policy is enforced based on the OpenFlow flows.



Conntrack integration

1. Traffic enters OVN.
2. The packet is sent to the conntrack module for processing.
3. The packet metadata is updated with state information.
4. The packet is returned to OVN with state information included in the metadata.

The Connection Tracker can contain multiple zones, which handle overlapping networks; for example, tenant networks. Each zone contains its own Connection Tracker table.

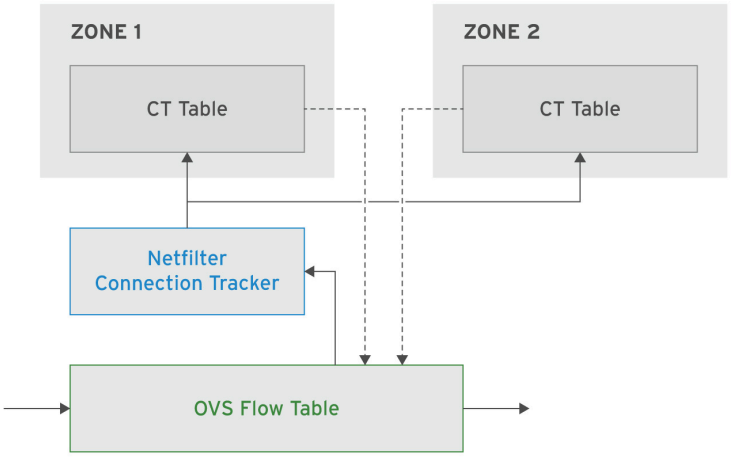


Figure 6.6: Multiple Conntrack Zones

Use the **ovn** commands to view the OpenFlow flows for security group implementation. Use the **ovn-vsctl list open** command to find the IP address for the OVN DB. Set the environment variables for the North and South databases.

```
[root@controller0 ~]# ovn-vsctl list open
_uuid               : e6cff4b4-5e7e-4a06-9d34-7060543b85b5
bridges             : [4d55bad8-8dbb-45a6-a059-291f5cfd4722, 57c81206-96be-48eb-
abac-22b71be9f07e, ac93ee97-e01f-4cde-ab1e-f8a6d10b0558]
cur_cfg             : 46
datapath_types      : [netdev, system]
db_version          : "7.15.1"
```

```
external_ids        : {hostname="controller0.overcloud.example.com", ovn-
bridge=br-int, ovn-bridge-mappings="datacentre:br-ex", ovn-cms-options=enable-
chassis-as-gw, ovn-encap-ip="172.24.2.1", ovn-encap-type=geneve, ovn-
remote="tcp:172.24.1.50:6642", rundir="/var/run/openvswitch", system-
id="ee00e7d6-3fb8-46cd-b2d7-2d3d3b92eb5b"}
iface_types         : [geneve, gre, internal, lisp, patch, stt, system, tap,
 vxlan]
manager_options     : []
next_cfg            : 46
other_config        : {}
ovs_version         : "2.9.0"
ssl                 : []
statistics          : {}
system_type         : rhel
system_version      : "7.5"
[root@controller0 ~]# export OVN_NB_DB=tcp:172.24.1.50:6641
[root@controller0 ~]# export OVN_SB_DB=tcp:172.24.1.50:6642
[root@controller0 ~]# env | grep OVN_
OVN_SB_DB=tcp:172.24.1.50:6642
OVN_NB_DB=tcp:172.24.1.50:6641
```

Use the **`ovn-nbctl`** to view the logical topology. Take notice of the ports listed, in particular `provider-datacentre`. The physical topology shows the logical ports bound to the OpenStack environment.

```
[root@controller0 ~]# ovn-nbctl show
...output omitted...
switch ce4e7293-9bed-4504-9272-1fc698393048 (neutron-
f8366a5d-9cf9-4d38-9760-9cf9df4eceaa) (aka provider-datacentre)
    port provnet-f8366a5d-9cf9-4d38-9760-9cf9df4eceaa
        type: localnet
        addresses: ["unknown"]
    port 059d906d-fb1d-4120-8c74-5cd9de12f832
        type: localport
        addresses: ["fa:16:3e:e6:57:f1"]
    port 8905aec6-0db1-475f-8e38-0dfd4b4fe4fc
        type: router
        router-port: lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc
    port 8e0a4263-3b98-49cc-81a9-499fec769040
        type: router
    router-port: lrp-8e0a4263-3b98-49cc-81a9-499fec769040
...output omitted...
```

List the OpenStack networks. Notice that the ID of `provider-datacentre` is the same in both outputs.

```
[user@demo ~]$ openstack network list -c ID -c Name
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 2fa7885c-c49d-452d-8501-3cb02582ebf4 | finance-network1    |
| f8366a5d-9cf9-4d38-9760-9cf9df4eceaa | provider-datacentre |
+--------------------------------------+---------------------+
```

To understand the logical flows for the implementation of security groups, list the OpenStack servers and OpenStack ports. Use the **`ovn-sbctl lflow-list`** command to list the OpenFlow rules. Find the OpenFlow listing for the **TCP** security rule.

```
[user@demo ~]$ openstack server list -c Name -c Networks
+-----------------+---------------------------------------------+
| Name            | Networks                                    |
+-----------------+---------------------------------------------+
| finance-server1 | finance-network1=192.168.1.6, 172.25.250.106 |
+-----------------+---------------------------------------------+
```

```
[user@demo ~]$ openstack port list -c ID -c "Fixed IP Addresses"
+-------------+-----------------------------------------------------+
| ID          | Fixed IP Addresses                                  |
+-------------+-----------------------------------------------------+
| 0625...f408 | ip_address='192.168.1.1', subnet_id='eab9...9590' |
| 0fcf...c2fa | ip_address='192.168.1.2', subnet_id='eab9...9590' |
| 79bb...70b6 | ip_address='192.168.1.6', subnet_id='eab9...9590' |
+-------------+-----------------------------------------------------+
```

```
[root@controller0 ~]# ovn-sbctl lflow-list
...output omitted...
table=4 (ls_out_acl         ), priority=2002 , match=(!ct.new && ct.est && !ct.rpl
 && ct_label.blocked == 0 && (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" &&
 ip4 && ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22)), action=(next;)
...output omitted...
table=4 (ls_out_acl         ), priority=2002 , match=(((ct.new && !ct.est)
 || (!ct.new && ct.est && !ct.rpl && ct_label.blocked == 1)) && (outport ==
 "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4 && ip4.src == 0.0.0.0/0 && tcp
 && tcp.dst == 22)), action=(reg0[1] = 1; next;)
...output omitted...
```

In the same output from the **`ovn-sbctl lflow-list`** command find the OpenFlow flows for the ICMP protocol.

```
[root@controller0 ~]# ovn-sbctl lflow-list
...output omitted...
table=4 (ls_out_acl         ), priority=2002 , match=(!ct.new && ct.est && !ct.rpl
 && ct_label.blocked == 0 && (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" &&
 ip4 && ip4.src == 0.0.0.0/0 && icmp4)), action=(next;)
...output omitted...
table=4 (ls_out_acl         ), priority=2002 , match=(((ct.new && !ct.est)
 || (!ct.new && ct.est && !ct.rpl && ct_label.blocked == 1)) && (outport ==
 "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4 && ip4.src == 0.0.0.0/0 && icmp4)),
 action=(reg0[1] = 1; next;)
...output omitted...
table=1 (lr_in_ip_input     ), priority=90   , match=(ip4.dst == 172.25.250.102
 && icmp4.type == 8 && icmp4.code == 0), action=(ip4.dst <-> ip4.src; ip.ttl =
 255; icmp4.type = 0; flags.loopback = 1; next; )
  table=1 (lr_in_ip_input     ), priority=90   , match=(ip4.dst == 192.168.1.1
 && icmp4.type == 8 && icmp4.code == 0), action=(ip4.dst <-> ip4.src; ip.ttl =
 255; icmp4.type = 0; flags.loopback = 1; next; )
...output omitted...
```

To see all of the security rules for a specific network, firstly use the **`ovn-nbctl ls-list`** command to list all existing switches. Find the ID of the network, then use the **`ovn-nbctl acl-list neutron-`*`2fa7885c-c49d-452d-8501-3cb02582ebf4`*** command to list the security rules.

```
[root@controller0 ~]# ovn-nbctl ls-list
...output omitted...
ac07...4e7c (neutron-2fa7885c-c49d-452d-8501-3cb02582ebf4)
ce4e...3048 (neutron-f8366a5d-9cf9-4d38-9760-9cf9df4eceaa)
...output omitted...
```

```
[user@demo ~]$ openstack network list -c ID -c Name
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 2fa7885c-c49d-452d-8501-3cb02582ebf4 | finance-network1    |
| f8366a5d-9cf9-4d38-9760-9cf9df4eceaa | provider-datacentre |
+--------------------------------------+---------------------+
```

```
[root@controller0 ~]# ovn-nbctl acl-list neutron-2fa7885c-
c49d-452d-8501-3cb02582ebf4
from-lport  1002 (inport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4) allow-
related
from-lport  1002 (inport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4 &&
 ip4.dst == {255.255.255.255, 192.168.1.0/24} && udp && udp.src == 68 && udp.dst
 == 67) allow
from-lport  1002 (inport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip6) allow-
related
from-lport  1001 (inport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip) drop
  to-lport  1002 (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4 &&
 ip4.src == $as_ip4_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1002 (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4 &&
 ip4.src == 0.0.0.0/0 && icmp4) allow-related
  to-lport  1002 (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22) allow-related
  to-lport  1002 (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip6 &&
 ip6.src == $as_ip6_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1001 (outport == "79bb40e8-343a-4d7b-810a-44412ca470b6" && ip) drop
```

NOTE

If there are no virtual machines attached to a network then the network will not contain any security rules. The rules are only applied when the first virtual machine is created. If a virtual machine is deleted from a network, OVN automatically deletes all rules for that virtual machine.

OVN DHCP

OVN implements DHCPv4 support which removes the need for a *DHCP agent*. Virtual networks no longer require a *DHCP namespace* or a *dnsmasq* process. To use the DHCP feature the `ovn_native_dhcp` parameter must be set to **`True`**. DHCPv4 options are configured on each compute node running `ovn-controller`. This means that DHCP support is fully distributed. DHCP requests from the VMs are also handled by `ovn-controller`.

The *DHCP_Options* table on the OVN Northbound DB stores the DHCP options. The database creates a new entry when a subnet is created if the **`enable_dhcp`** option is set to **`True`**. DHCPv4 options stored in the *options* column include the reply sent by `ovn-controller` when it receives a DHCPv4 request.

The `ovn-northd` service adds the logical flows for each logical port where DHCP options are defined. It adds two stages to the ingress pipeline: *ls_in_dhcp_options* and *ls_in_dhcp_response*. Logical flows exist for each individual instance. The `put_dhcp_opts` action transforms the DHCP request into a reply. An invalid packet remains unchanged by the `put_dhcp_opts` OVN action.

```
[root@controller0 ~]# ovn-sbctl lflow-list
...output omitted
table=12(ls_in_dhcp_options ), priority=100  , match=(inport ==
 "79bb40e8-343a-4d7b-810a-44412ca470b6" && eth.src == fa:16:3e:8b:61:a5 &&
 ip4.src == 192.168.1.6 && ip4.dst == {192.168.1.1, 255.255.255.255} && udp.src
 == 68 && udp.dst == 67), action=(reg0[3] = put_dhcp_opts(offerip = 192.168.1.6,
 classless_static_route = {169.254.169.254/32,192.168.1.2, 0.0.0.0/0,192.168.1.1},
 dns_server = {172.25.250.254}, lease_time = 43200, mtu = 1442, netmask =
 255.255.255.0, router = 192.168.1.1, server_id = 192.168.1.1); next;)
...output omitted
table=13(ls_in_dhcp_response), priority=100  , match=(inport ==
 "79bb40e8-343a-4d7b-810a-44412ca470b6" && eth.src == fa:16:3e:8b:61:a5 && ip4 &&
 udp.src == 68 && udp.dst == 67 && reg0[3]), action=(eth.dst = eth.src; eth.src
 = fa:16:3e:4a:55:7e; ip4.dst = 192.168.1.6; ip4.src = 192.168.1.1; udp.src = 67;
 udp.dst = 68; outport = inport; flags.loopback = 1; output;)
...output omitted
```

ovs-vswitchd sends a DHCP request packet to the `ovn-controller`. The `ovn-controller` extracts the IP and DHCP options, frames the reply, and sends the packet back to the switch. On reception of the packet, `ovs-vswitchd` executes the next pipeline: *ls_in_dhcp_response*. The `flags.loopback = 1` flow sends the DHCP packet to the VM port.

OVN LOAD BALANCING

The *OVN load balancer* provides basic load balancing services. It uses a hash-based algorithm to balance requests. The balancing looks random because the hash is calculated using the headers of the client request. An individual client's requests are handled by the same load balancer for the duration of the connection. The load balancing is applied either to a logical switch or a logical router.

If the OVN load balancer is applied to a logical router then that router must be a gateway router. This means that load balancing is a centralized service.

If the OVN load balancer is applied to a logical switch then load balancing is distributed because it is applied to multiple OVS hosts. Only ingress traffic from a VIF is evaluated in this case, which means that it must be applied on the client logical switch. In a large-scale environment the OVN load balancer must be applied to many logical switches. The OVN load balancer is also implemented using OpenFlow flows.

OVN METADATA

A metadata agent called `neutron-ovn-metadata-agent` runs on each controller node. This agent starts the HAProxy instances. The HAProxy instances manage the OVS interfaces, network namespaces, and HAProxy processes. The `neutron-ovn-metadata-agent` agent is also the connection between HAProxy and the Nova metadata API service. HAProxy adds headers to the metadata API request and forwards it to the `neutron-ovn-metadata-agent` agent.

The instances reach the metadata namespace through a static route. The instances direct traffic to `169.254.169.254`, but the traffic is redirected to the IP address of the metadata port for that network.

The `neutron-ovn-metadata-agent` agent ensures that the metadata proxies are running. It also watches the Port-Binding table of the Southbound database. For entries where the *chassis* column is set to a host where an agent is running, the `neutron-ovn-metadata-agent` agent ensures a metadata proxy instance is running for the neutron network data paths. A record is kept of all networks where proxies are running. This is done by updating the `neutron-metadata-proxy-networks` key in the Southbound database. Any metadata proxies that are no longer required are automatically removed by the `neutron-ovn-metadata-agent`.

The agent maintains a connection to the Northbound database. When a new network is created a new OVN local port is also created. This local port acts as a metadata proxy. When the network is removed the local port is also removed. If the DHCP setting is set to **`False`** the metadata port is created anyway if the metadata service is enabled.

Further information on the metadata service can be found in the Initializing an Instance During Deployment section of this book.

REFERENCES

Further information is available in the *Networking with Open Virtual Network* for Red Hat OpenStack Platform at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/

IMPLEMENTING OPEN VIRTUAL NETWORKS (OVN)

In this exercise, you will manage OVN and security groups, look at OpenFlow flows, and review the DHCP configuration and the routing tables.

OUTCOMES

You should be able to:

- Deploy an instance with public access.
- Create security group rules and review OpenFlow flows for security policies.
- Review the OpenFlow rules for DHCP.
- Review the routing tables in OpenFlow.

BEFORE YOU BEGIN

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

Run the **`lab networking-ovn setup`** command. This script ensures that OpenStack services are running and the environment is properly configured for the exercise.

```
[student@workstation ~]$ lab networking-ovn setup
```

1. As the `developer1` user, confirm that the `finance-network1` network exists and review the security group rules for the network.

   1.1. On `workstation`, source the **`/home/student/developer1-rc`** identity environment file to provide access to OpenStack services for the `developer1` user.

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

   1.2. Use the **`openstack network list`** command to list the available networks. Note the ID of `finance-network1`.

```
[student@workstation ~(developer1-finance)]$ openstack network list -c ID -c Name
+--------------------------------------+---------------------+
| ID                                   | Name                |
+--------------------------------------+---------------------+
| 7ad21970-5986-42e0-9b48-64ab414bb0b1 | finance-network1    |
| d55f6d1e-c29e-4825-8de4-01dd95f8a220 | provider-storage    |
| f8366a5d-9cf9-4d38-9760-9cf9df4eceaa | provider-datacentre |
+--------------------------------------+---------------------+
```

   1.3. Open another terminal. From `workstation`, log in to the `controller0` node as the `heat-admin` user. Use the **`sudo -i`** command to gain root privileges.

```
[student@workstation ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

1.4. On `controller0`, use the **`ovs-vsctl list open`** command to find the IP address of the Northbound and Southbound databases. The IP address is in the `ovn-remote` parameter. Set the environment variables for the OVN databases.

```
[root@controller0 ~]# ovs-vsctl list open
...output omitted...
external_ids        : {hostname="controller0.overcloud.example.com", ovn-
bridge=br-int, ovn-bridge-mappings="datacentre:br-ex", ovn-cms-options=enable-
chassis-as-gw, ovn-encap-ip="172.24.2.1", ovn-encap-type=geneve, ovn-
remote="tcp:172.24.1.50:6642", rundir="/var/run/openvswitch", system-id="698c1e91-
b2c0-455a-becb-0baa27d275ad"}
...output omitted...
```

```
[root@controller0 ~]# export OVN_SB_DB=tcp:172.24.1.50:6642
[root@controller0 ~]# export OVN_NB_DB=tcp:172.24.1.50:6641
```

1.5. Use the **`ovn-nbctl ls-list`** command to list all existing switches. Using the **`ovn-nbctl acl-list neutron-`*`switch_ID`*** command, list the security rules for the network switch. The `switch_ID` matches the ID of the `finance-network1` network. Notice that there are currently no security rules assigned to the network switch.

```
[root@controller0 ~]# ovn-nbctl ls-list
...output omitted...
bed0...4def (neutron-7ad21970-5986-42e0-9b48-64ab414bb0b1)
...output omitted...
```

```
[root@controller0 ~]# ovn-nbctl acl-list \
neutron-7ad21970-5986-42e0-9b48-64ab414bb0b1
[root@controller0 ~]#
```

2. On the `workstation` terminal, as the `developer1` user, create a new instance called `finance-server1`. Use the `rhel7` image, the `default` flavor, the `default` security group, and the `finance-network1` network.

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--nic net-id=finance-network1 \
--security-group default \
--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       | 2018-09-26T13:33:49.000000       |
| OS-SRV-USG:terminated_at     | None                             |
| accessIPv4                   |                                  |
| accessIPv6                   |                                  |
| addresses                    | finance-network1=192.168.1.10    |
| adminPass                    | hyP4Dn3Dvse6                     |
| config_drive                 |                                  |
| created                      | 2018-09-26T13:33:14Z             |
| flavor                       | default (71eb...a769)            |
| hostId                       | deb5...932f                      |
| id                           | e417...5720                      |
| image                        | rhel7 (ec201...e840)             |
| key_name                     | None                             |
| name                         | finance-server1                  |
| progress                     | 0                                |
| project_id                   | 0baea8e7fc5b4720bc418c0172ee6ae5|
| properties                   |                                  |
| security_groups              | name='default'                   |
| status                       | ACTIVE                           |
| updated                      | 2018-09-26T13:33:49Z             |
| user_id                      | 92f5297388884088b72f01b0f2787127|
| volumes_attached             |                                  |
+-----------------------------+----------------------------------+
```

3. On `controller0`, recheck the security policies for the `finance-network1` network switch. On `workstation`, list the security group rules for the `default` security group. Match the OpenFlow flows to the security group rules. Match the **`inport`** and **`outport`** parameters to their respective OpenStack entity.

   3.1. On `controller0` use the **`ovn-nbctl acl-list neutron-`*`switch_ID`*** command to recheck the security group policies for the `finance-network1` network switch.

```
[root@controller0 ~]# ovn-nbctl acl-list \
neutron-7ad21970-5986-42e0-9b48-64ab414bb0b1
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4) allow-
related
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.dst == {255.255.255.255, 192.168.1.0/24} && udp && udp.src == 68 && udp.dst
 == 67) allow
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip6) allow-
related
from-lport  1001 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip) drop
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == $as_ip4_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && icmp4) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 443) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 80) allow-related
```

```
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip6 &&
 ip6.src == $as_ip6_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1001 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip) drop
```

3.2. On `workstation`, use the **`openstack security group show default`** command to show the configuration of the `default` security group. Notice that the OpenFlow flows in the previous step match the security group rules in OpenStack. Also notice that the last action is `drop`.

```
[student@workstation ~(developer1-finance)]$ openstack security group \
rule list default --long -f json
[
  {
    "IP Range": "0.0.0.0/0",
    "Direction": "ingress",
    "Ethertype": "IPv4",
    "Port Range": "22:22",
    "IP Protocol": "tcp",
    "Remote Security Group": null,
    "ID": "b67c8063-28ac-4dc2-a734-3cb6b2de2e42"
  },
  {
    "IP Range": null,
    "Direction": "egress",
    "Ethertype": "IPv4",
    "Port Range": "",
    "IP Protocol": null,
    "Remote Security Group": null,
    "ID": "bbc68ba4-89ea-4b34-8ca7-8e8cd5cc2a4f"
  },
  {
    "IP Range": null,
    "Direction": "ingress",
    "Ethertype": "IPv4",
    "Port Range": "",
    "IP Protocol": null,
    "Remote Security Group": "feef8ea8-b579-4e8f-bda3-b880af1a468e",
    "ID": "717925c3-fb1f-4e87-b8f2-4b6a0127d648"
  },
  {
    "IP Range": "0.0.0.0/0",
    "Direction": "ingress",
    "Ethertype": "IPv4",
    "Port Range": "443:443",
    "IP Protocol": "tcp",
    "Remote Security Group": null,
    "ID": "6b8aa744-cd3a-449f-bdca-dc75d70b1621"
  },
  {
    "IP Range": "0.0.0.0/0",
    "Direction": "ingress",
    "Ethertype": "IPv4",
    "Port Range": "80:80",
    "IP Protocol": "tcp",
    "Remote Security Group": null,
    "ID": "6de5bdd1-dc72-4c1c-af07-cd900c96ffa8"
```

```
  },
  {
    "IP Range": null,
    "Direction": "ingress",
    "Ethertype": "IPv6",
    "Port Range": "",
    "IP Protocol": null,
    "Remote Security Group": "feef8ea8-b579-4e8f-bda3-b880af1a468e",
    "ID": "361e6de4-c1b7-47e3-887d-0ff475464c05"
  },
  {
    "IP Range": null,
    "Direction": "egress",
    "Ethertype": "IPv6",
    "Port Range": "",
    "IP Protocol": null,
    "Remote Security Group": null,
    "ID": "8659559d-4024-4083-8cda-38acf9d3f97e"
  },
  {
    "IP Range": "0.0.0.0/0",
    "Direction": "ingress",
    "Ethertype": "IPv4",
    "Port Range": "",
    "IP Protocol": "icmp",
    "Remote Security Group": null,
    "ID": "cdf56956-a6f2-4ba5-8974-eb5c60afd82f"
  }
]
```

3.3. On `workstation`, use the **`openstack port list`** command to list the OpenStack ports. Notice that the port for the `finance-server1` instance matches the **`inport`** and the **`outport`** parameters.

```
[student@workstation ~(developer1-finance)]$ openstack port list \
-c ID -c "Fixed IP Addresses"
+-------------+-----------------------------------------------------+
| ID          | Fixed IP Addresses                                  |
+-------------+-----------------------------------------------------+
| 12c7...00d5 | ip_address='192.168.1.1', subnet_id='849b...9ab9'   |
| a557...1106a| ip_address='192.168.1.10', subnet_id='849...79ab9' |
| e56f...2b89 | ip_address='192.168.1.2', subnet_id='849b...9ab9'   |
+-------------+-----------------------------------------------------+
```

```
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4) allow-
related
   from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.dst == {255.255.255.255, 192.168.1.0/24} && udp && udp.src == 68 && udp.dst
 == 67) allow
...output omitted...
to-lport 1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22) allow-related
```

4. Create a second instance on the `finance-network1` network. On the `controller0` node, list the security rule flows.

4.1. On `workstation`, as the `developer1` user, create a new instance called `finance-server2`. Use the `rhel7` image, the `default` flavor, and the `finance-network1` network. Notice that although no security group was defined, the `default` security group was assigned to the instance.

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--nic net-id=finance-network1 \
--wait finance-server2
+-----------------------------+---------------------------------+
| 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      | 2018-09-26T17:15:05.000000      |
| OS-SRV-USG:terminated_at    | None                            |
| accessIPv4                  |                                 |
| accessIPv6                  |                                 |
| addresses                   | finance-network1=192.168.1.12   |
| adminPass                   | KJ6jneM5sxeW                    |
| config_drive                |                                 |
| created                     | 2018-09-26T17:14:18Z            |
| flavor                      | default (71eb...a769)           |
| hostId                      | 445c..81cc                      |
| id                          | 4dfa...e4b9                     |
| image                       | rhel7 (ec20...e840)             |
| key_name                    | None                            |
| name                        | finance-server2                 |
| progress                    | 0                               |
| project_id                  | 0baea8e7fc5b4720bc418c0172ee6ae5|
| properties                  |                                 |
| security_groups             | name='default'                  |
| status                      | ACTIVE                          |
| updated                     | 2018-09-26T17:15:05Z            |
| user_id                     | 92f5297388884088b72f01b0f2787127|
| volumes_attached            |                                 |
+-----------------------------+---------------------------------+
```

4.2. On `controller0` check the security policies for the `finance-network` network switch. Notice the new OpenFlow flows. Also notice that the last rule for each instance is the **`drop`** rule.

```
[root@controller0 ~]# ovn-nbctl acl-list \
neutron-7ad21970-5986-42e0-9b48-64ab414bb0b1
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4) allow-
related
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.dst == {255.255.255.255, 192.168.1.0/24} && udp && udp.src == 68 && udp.dst
 == 67) allow
```

```
from-lport  1002 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip6) allow-
related
from-lport  1002 (inport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4) allow-
related
from-lport  1002 (inport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4 &&
 ip4.dst == {255.255.255.255, 192.168.1.0/24} && udp && udp.src == 68 && udp.dst
 == 67) allow
from-lport  1002 (inport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip6) allow-
related
from-lport  1001 (inport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip) drop
from-lport  1001 (inport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip) drop
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == $as_ip4_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && icmp4) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 443) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 80) allow-related
  to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip6 &&
 ip6.src == $as_ip6_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1002 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4 &&
 ip4.src == $as_ip4_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1002 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4 &&
 ip4.src == 0.0.0.0/0 && icmp4) allow-related
  to-lport  1002 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22) allow-related
  to-lport  1002 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 443) allow-related
  to-lport  1002 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 80) allow-related
  to-lport  1002 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip6 &&
 ip6.src == $as_ip6_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
  to-lport  1001 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip) drop
  to-lport  1001 (outport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip) drop
```

4.3. On `workstation`, list the OpenStack ports. Look at the security group OpenFlow flows again, and notice the flows allowing traffic to flow between the instances. In the following example, the **`inport`** is the new instance `finance-server2`. The **`outport`** is the instance created previously, `finance-server1`. The two rules are **`icmp`** and **`tcp`**.

```
[student@workstation ~(developer1-finance)]$ openstack port list \
-c ID -c "Fixed IP Addresses"
+-------------+-------------------------------------------------+
| ID          | Fixed IP Addresses                              |
+-------------+-------------------------------------------------+
| 12c7...00d5 | ip_address='192.168.1.1', subnet_id='849b...9ab9'  |
| a557...106a | ip_address='192.168.1.10', subnet_id='849b...9ab9' |
| b3e3...7702 | ip_address='192.168.1.12', subnet_id='849b...9ab9' |
| e56f...2b89 | ip_address='192.168.1.2', subnet_id='849b...9ab9'  |
```

```
+-------------+-----------------------------------------------------+
```

```
from-lport  1001 (inport == "b3e37780-7fe3-471a-bcdd-73bdcb957702" && ip) drop
    to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-
d343d3d1106a" && ip4 && ip4.src ==
 $as_ip4_feef8ea8_b579_4e8f_bda3_b880af1a468e) allow-related
    to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && icmp4) allow-related
    to-lport  1002 (outport == "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && ip4 &&
 ip4.src == 0.0.0.0/0 && tcp && tcp.dst == 22) allow-related
```

5. On `workstation`, create a floating IP address on the `provider-datacentre` external network. Attach the floating IP to `finance-server1`. Use the **`openstack server list`** command to verify the addition of the floating IP address.

```
[student@workstation ~(developer1-finance)]$ openstack floating ip create \
provider-datacentre
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| created_at          | 2018-09-27T08:22:32Z                 |
| description         |                                      |
| fixed_ip_address    | None                                 |
| floating_ip_address | 172.25.250.108                       |
| floating_network_id | f8366a5d-9cf9-4d38-9760-9cf9df4eceaa |
| id                  | c4416eaa-35a3-47e6-9546-e8256981c069 |
| name                | 172.25.250.110                       |
| port_id             | None                                 |
| project_id          | 0baea8e7fc5b4720bc418c0172ee6ae5     |
| qos_policy_id       | None                                 |
| revision_number     | 0                                    |
| router_id           | None                                 |
| status              | DOWN                                 |
| subnet_id           | None                                 |
| updated_at          | 2018-09-27T08:22:32Z                 |
+---------------------+--------------------------------------+
```

```
[student@workstation ~(developer1-finance)]$ openstack server add \
floating ip finance-server1 172.25.250.108
[student@workstation ~(developer1-finance)]$ openstack server list \
-c ID -c Name -c Networks
+-------------+-----------------+---------------------------------------------+
| ID          | Name            | Networks                                    |
+-------------+-----------------+---------------------------------------------+
| 4dfa...e4b9 | finance-server2 | finance-network1=192.168.1.12               |
| e417...5720 | finance-server1 | finance-network1=192.168.1.10, 172.25.250.108 |
+-------------+-----------------+---------------------------------------------+
```

6. On `controller0`, list the OpenFlow flows for routing and look at the routing table for the `finance-network1` network switch. On `controller0`, use the **`ovn-nbctl show`** command to show a brief overview of the contents of the Northbound database.

   6.1. On `controller0`, use the **`ovn-sbctl lflow-list`** command to list the flows. Find the **SNAT** and **DNAT** flows for `finance-server1`. The easiest way to find the routing flows is to redirect the output of the **`ovn-sbctl`** command to a file. Open the file and use the search functionality. Notice that there are OpenFlow rules for both the floating IP address and the internal IP address.

```
[root@controller0 ~]# ovn-sbctl lflow-list > flow.txt
[root@controller0 ~]# view flow.txt
```

```
...output omitted...
  table=3 (lr_in_unsnat       ), priority=100  , match=(ip && ip4.dst
 == 172.25.250.108 && inport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"),
 action=(ct_snat;)
...output omitted...
  table=3 (lr_in_unsnat       ), priority=50   , match=(ip && ip4.dst
 == 172.25.250.108), action=(reg9[0] = 1; next;)
...output omitted
  table=1 (lr_out_snat        ), priority=33   , match=(ip && ip4.src
 == 192.168.1.10 && outport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"),
 action=(eth.src = fa:16:3e:76:a8:89; ct_snat(172.25.250.108);)
...output omitted...
  table=4 (lr_in_dnat         ), priority=100  , match=(ip && ip4.dst
 == 172.25.250.108 && inport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"),
 action=(ct_dnat(192.168.1.10);)
  table=4 (lr_in_dnat         ), priority=50   , match=(ip && ip4.dst
 == 172.25.250.108), action=(reg9[0] = 1; next;)
  table=4 (lr_in_dnat         ), priority=0    , match=(1), action=(next;)
...output omitted...
  table=0 (lr_out_undnat      ), priority=100  , match=(ip && ip4.src
 == 192.168.1.10 && outport == "lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc"),
 action=(eth.src = fa:16:3e:76:a8:89; ct_dnat;)
...output omitted...
```

   6.2. From the SNAT or DNAT extract used in the previous step, note the **`lrp`** inport and outport used by `finance-server1`. On `controller0`, use the **`ovn-nbctl show`** command to list the contents of the Northbound database.

```
[root@controller0 ~]# ovn-nbctl show
...output omitted...
switch ce4e7293-9bed-4504-9272-1fc698393048 (neutron-
f8366a5d-9cf9-4d38-9760-9cf9df4eceaa) (aka provider-datacentre)
  port provnet-f8366a5d-9cf9-4d38-9760-9cf9df4eceaa
     type: localnet
     addresses: ["unknown"]
  port 059d906d-fb1d-4120-8c74-5cd9de12f832
     type: localport
     addresses: ["fa:16:3e:e6:57:f1"]
  port 8905aec6-0db1-475f-8e38-0dfd4b4fe4fc
     type: router
     router-port: lrp-8905aec6-0db1-475f-8e38-0dfd4b4fe4fc
  port 8e0a4263-3b98-49cc-81a9-499fec769040
```

```
    type: router
    router-port: lrp-8e0a4263-3b98-49cc-81a9-499fec769040
...output omitted...
```

Notice that the **lrp** inport and outport are router ports for the `provider-datacentre` external network.

7. Show the routing table for `finance-router1`.

7.1. On `workstation`, use the **`openstack router list`** command to list the OpenStack routers.

```
[student@workstation ~(developer1-finance)]$ openstack router list -c ID -c Name
+--------------------------------------+-----------------+
| ID                                   | Name            |
+--------------------------------------+-----------------+
| 54a04000-17ea-44df-8c79-d18164e6a1f2 | finance-router1 |
+--------------------------------------+-----------------+
```

7.2. On `controller0`, use the **`ovn-nbctl lr-list`** command to list all existing routers.

```
[root@controller0 ~]# ovn-nbctl lr-list
df2df96e-d19f-48e9-8204-391e22b35efd (neutron-54a04000-17ea-44df-8c79-
d18164e6a1f2)
1baf78f6-b8cc-468c-bd7e-4f01d363c5b5 (neutron-6e39ddfe-1a26-46f7-
b602-15373b615a1e)
```

7.3. On `controller0`, use the **`ovn-nbctl lr-nat-list`** command to show the routing table for `finance-router1`.

```
[root@controller0 ~]# ovn-nbctl lr-nat-list \
neutron-54a04000-17ea-44df-8c79-d18164e6a1f2
TYPE            EXTERNAL_IP     LOGICAL_IP    EXTERNAL_MAC       LOGICAL_PORT
dnat_and_snat   172.25.250.108 192.168.1.10 fa:16:3e:76:a8:89 a557...106a
snat            172.25.250.102 192.168.1.0/24
```

7.4. On `workstation`, use the **`openstack port list`** command to list the OpenStack ports.

Look at the routing table in the previous step. Notice that the OpenStack port ID for IP address `192.168.1.10` matches the **`Logical Port`** option in the OVN routing table. Also notice that there are no SNAT or DNAT rules for IP address `192.168.1.12`. `finance-server2` has no floating IP address attached to it, and therefore there are no routing flows for that instance.

```
[student@workstation ~(developer1-finance)]$ openstack port list \
-c ID -c "Fixed IP Addresses"
+-------------+-----------------------------------------------------+
| ID          | Fixed IP Addresses                                  |
+-------------+-----------------------------------------------------+
| 12c7...00d5 | ip_address='192.168.1.1', subnet_id='849b...9ab9'   |
| a557...106a | ip_address='192.168.1.10', subnet_id='849b...9ab9'  |
| b3e3...7702 | ip_address='192.168.1.12', subnet_id='849b...9ab9'  |
| e56f...2b89 | ip_address='192.168.1.2', subnet_id='849b...9ab9'   |
+-------------+-----------------------------------------------------+
```

8. On `controller0`, use the **`ovn-sbctl`** command to list the logical flows. Redirect the output to a file called **`dhcp.txt`** In the **`dhcp.txt`** file, find the **`ls_in_dhcp_options`**, the **`ls_in_dhcp_response`**, and **`put_dhcp_opts`** options for the `finance-server1`.

```
[root@controller0 ~]# ovn-sbctl lflow-list > dhcp.txt
[root@controller0 ~]# view dhcp.txt

table=12(ls_in_dhcp_options ), priority=100  , match=(inport ==
 "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && eth.src == fa:16:3e:eb:a3:bc && ip4.src
 == 0.0.0.0 && ip4.dst == 255.255.255.255 && udp.src == 68 && udp.dst == 67),
 action=(reg0[3] = put_dhcp_opts(offerip = 192.168.1.10, classless_static_route
 = {169.254.169.254/32,192.168.1.2, 0.0.0.0/0,192.168.1.1}, dns_server =
 {172.25.250.254}, lease_time = 43200, mtu = 1442, netmask = 255.255.255.0, router
 = 192.168.1.1, server_id = 192.168.1.1); next;)

  table=13(ls_in_dhcp_response), priority=100  , match=(inport ==
 "a557dbb8-3d1b-4780-9ca7-d343d3d1106a" && eth.src == fa:16:3e:eb:a3:bc && ip4 &&
 udp.src == 68 && udp.dst == 67 && reg0[3]), action=(eth.dst = eth.src; eth.src =
 fa:16:3e:d0:10:a8; ip4.dst = 192.168.1.10; ip4.src = 192.168.1.1; udp.src = 67;
 udp.dst = 68; outport = inport; flags.loopback = 1; output;)
```

9. Close the terminal windows used in this exercise.

Cleanup

On `workstation`, run the **`lab networking-ovn cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab networking-ovn cleanup
```

This concludes the guided exercise.

SELECTING NETWORK PROVISIONING OPTIONS

OBJECTIVES

After completing this section, students should be able to:

- Compare self-service and provider networks.
- Assign an IP address from the DHCP server to a subnet.

SELF-SERVICE (TENANT) NETWORKS

Self-service networks contain only virtual components. Projects are not concerned with the underlying physical network because tenant networks are isolated by default. Tenant networks can be connected to each other or to an external network through a router if required.

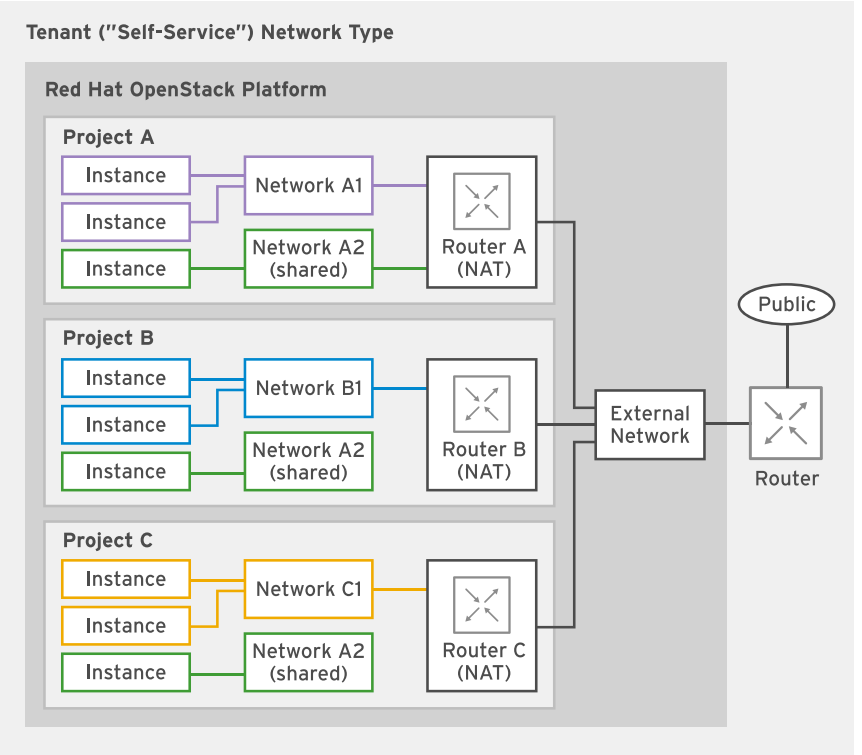


Figure 6.7: The self-service tenant networking scenario

When a new VLAN-enabled network is created, either as a tenant network by a self-service user or a provider network by an administrator, a VLAN ID is assigned from the configured **`network_vlan_ranges`** values and set as the `provider:segmentation_id` value. The `segmentation_id` field is used by all network types (VLAN, Geneve) to hold their protocol ID. Use the **`openstack network show`** command to view this parameter for a given network, regardless of the protocol displayed in the `network_type` field.

```
[demo@workstation ~]$ openstack network show 6fce40cf-7da3-4242-938a-3c7043dc2b6a
+---------------------------+------------------------+
| Field                     | Value                  |
+---------------------------+------------------------+
```

```
...output omitted...
| name                      | research-vlan          |
| provider:network_type     | vlan                   |
| provider:physical_network | datacenter             |
| provider:segmentation_id  | 4001                   |
...output omitted...
```

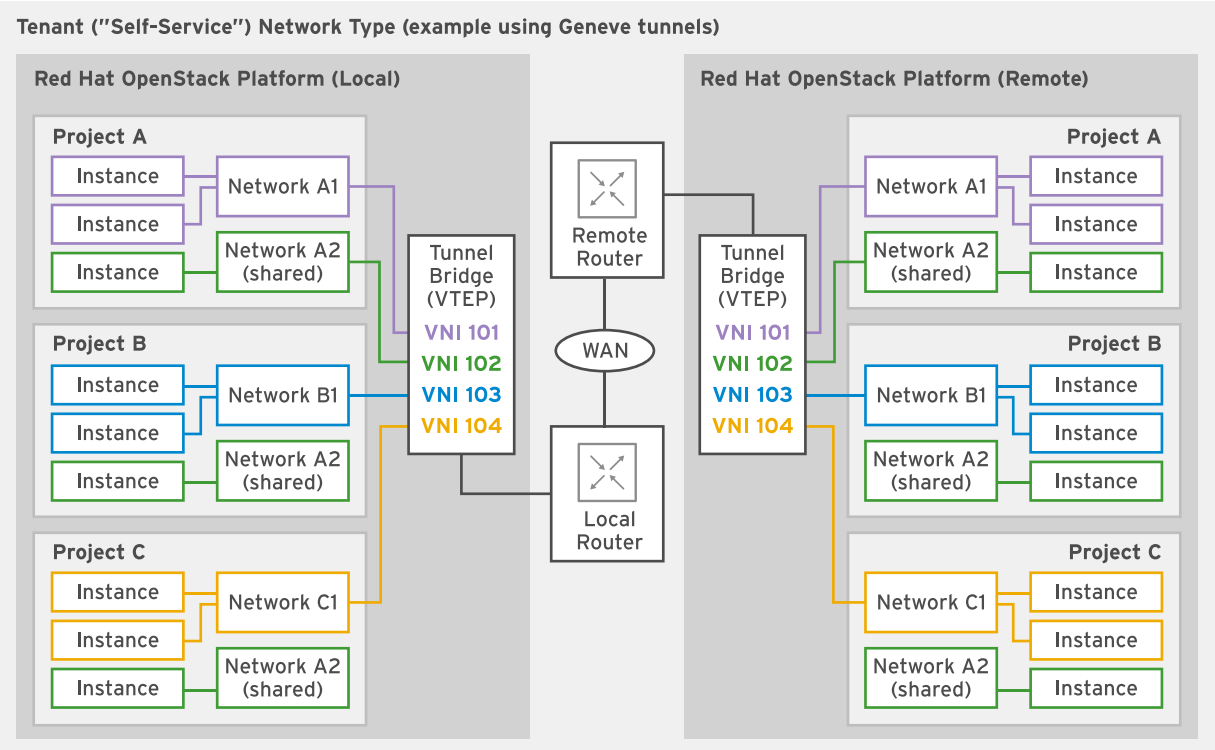


Figure 6.8: The self-service tenant networking scenario using tunneling to remote overcloud nodes.

PROVIDER (DATA CENTER) NETWORKS

Provider networks are typically created on physical network VLANs in the data center, but they can also be built using overlay protocols. The default overlay protocol in Red Hat OpenStack is Geneve. Overlay networks are routed to the rest of the physical network, similar to how existing tenant networks are built. Tenant networks allow address scope overlap by using routers and NAT tables to isolate the internal fixed addresses from external floating IP addresses.

Provider networks eliminate the routers, floating IPs, and self-service fixed IP addresses of tenant networks. Instead, provider networks use addresses taken directly from the physical network infrastructure, and rely on the physical network routers to operate as their default gateway.

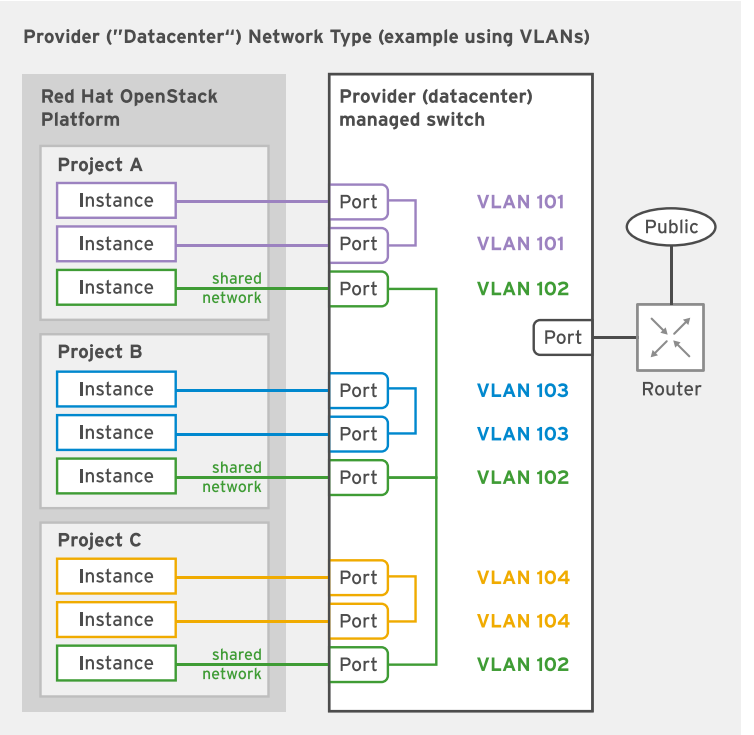


Figure 6.9: The provider networking scenario using a data center-provided hardware switch with VLANs.

In this classroom, there are two bridges mapped for the use of provider networks. The interfaces `eth3` and `eth4` are each bound to the bridges `br-eth3` and `br-eth4`. These bridges are mapped to the names `vlanprovider1` and `vlanprovider2`, with allowed VLAN ranges from 101 to 104 inclusive. This means that a provider network created using the `vlanprovider1` mapping will send traffic out through the `eth3` interface on a compute node.

COMPARING TENANT NETWORKS WITH PROVIDER NETWORKS

There are several differences between tenant and provider networks.

Creating Different Network Types

One of the differences between tenant and provider networks is who has the ability to create them. Provider networks can only be created by OpenStack administrators, and they are then shared with tenant projects. Self-service, or tenant networks, are created by tenants for use within their project and are not shared between projects unless configured by an OpenStack administrator.

Network Connection Types

Attaching an instance to a provider network allows the instances to have a layer 2 connection to the upstream networks in the organization. Provider networks are commonly created for specific VLANs and use 802.1q tagging, however flat network segments can also be used.

An instance attached to a tenant network can only connect to the organization's upstream networks if the tenant network is attached to an OpenStack router that has an external network connected. An external network is a special provider network that is designed for routed connections, not for directly attaching instances. Because tenant networks can use any IP addressing scheme, when connected to an SDN router with an external network connection, Network Address Translation is performed on all outbound traffic.

To allow inbound traffic to an instance attached to a tenant network, you need to allocate a floating IP address, which is then hosted on an OpenStack router to provide Static NAT.

Centralized and Distributed Routers

Before OpenStack introduced Distributed Virtual Routers (DVR), all tenant project traffic was forwarded to the software-defined Layer 3 devices configured on controller or network nodes, which increased traffic latency and decreased network performance.

When using provider networks, traffic from the instance will leave the compute node directly for the upstream network. Traffic leaving an instance that is attached to a tenant network must pass through an OpenStack router before reaching the upstream network.

Choosing Between Self-service and Provider Networks

The following list describes some use cases for self-service and provider networks.

- When the OpenStack platform is used as a general purpose computing pool by several projects or business units within the organization, self-service networks can reduce the burden on administrators. A pool of floating IP addresses can be created up front and allocated on demand by tenants.
- If the OpenStack platform is used for a specific purpose or application, the use of provider networks may only require some initial configuration with fewer ongoing changes. An external firewall can be used to segregate an OpenStack hosted application into tiers when VLAN provider networks are used.
- The use of provider networks can increase network performance because there is no router between the instances and the upstream network equipment. Traffic from the instance will leave directly from the compute node without transiting a controller or network node.

SUBNET POOLS

OpenStack was designed for multitenancy. Individual projects can be privately managed, only allowing access to internal systems through the use of NAT and a well-managed pool of floating IP addresses. Overlapping addresses in subnets is managed by OVN.

Provider networks require more stringent management of IP addresses. These networks are not routed to external networks through NAT; instead, they are directly on the external or internal networks. They use existing IP address ranges already configured on the physical switches, routers, and systems. IP addresses are automatically allocated from the configured allocation pool.

```
[user@demo ~]$ openstack subnet show provider-subnet-172.25.250
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 172.25.250.101-172.25.250.189        |
...output omitted...
```

To assign a specific IP address to an instance you need to know which IP addresses are still free. You need admin privileges to see the entire IP list so that you can choose one that is available.

When creating an OpenStack subnet pool the user chooses the subnet range.

Subnet pools track ranges already allocated, ensuring that addresses never overlap. If the subnet address pool was built using externally routable address scopes, then all projects using this subnet pool will have addresses that are routable and unique. This is especially useful for IPv6, which uses globally unique and routable addresses without the floating IP technique.

Subnet pools can be shared, which allows them to be used by subnets from multiple projects. For IPv4, subnets created from the same subnet pool and connected to the same routers can communicate between those networks without using floating IP addresses or traversing external networks or routers. Project members can create subnet pools visible only to their project. Shared subnet pools require admin privileges.

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/

SELECTING NETWORK PROVISIONING OPTIONS

In this exercise, you will create a provider network, and then verify the components involved.

OUTCOMES

You should be able to:

- Create a VLAN provider network.
- Describe the components and configuration required to support a VLAN provider network.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab networking-provisioning setup`** command. This script creates any resources required to perform the exercise.

```
[student@workstation ~]$ lab networking-provisioning setup
```

1. Log in to `controller0` and determine the available physical networks where you can create provider networks.

   1.1. From `workstation`, log in to `controller0` and change to the `root` user.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

   1.2. Use the **`ovs-vsctl`** command to display the **`ovn-bridge-mappings`** configuration. This output lists which bridges are mapped to physical interfaces and are available for use as provider networks.

```
[root@controller0 ~]# ovs-vsctl get open . external-ids:ovn-bridge-mappings
"datacentre:br-ex,vlanprovider1:br-eth3,vlanprovider2:br-eth4,storage:br-trunk"
```

   The `datacentre` mapping is already in use to provide the flat external network. The `storage` mapping allows instances to mount CephFS volumes. This leaves `vlanprovider1` and `vlanprovider2` available for provider networks.

   1.3. Use the **`ovs-vsctl`** command to display the `br-eth3` and `br-eth4` bridges.

```
[root@controller0 ~]# ovs-vsctl show
...output omitted...
    Bridge "br-eth4"
        fail_mode: standalone
        Port "eth4"
            Interface "eth4"
```

```
        Port "br-eth4"
            Interface "br-eth4"
                type: internal
    Bridge "br-eth3"
        fail_mode: standalone
        Port "eth3"
            Interface "eth3"
        Port "br-eth3"
            Interface "br-eth3"
                type: internal
...output omitted...
```

Note that these bridges have no connection other than the physical interfaces.

1.4. Determine whether VLAN ID ranges have been defined for any of the bridge mappings, then log out of `controller0`.

```
[root@controller0 ~]# grep ^network_vlan_ranges \
/var/lib/config-data/puppet-generated/neutron/etc/neutron/plugins/ml2/ml2_conf.ini
network_vlan_ranges=datacentre:1:1000,vlanprovider1:101:104,vlanprovider2:101:104,
storage:30:30
[root@controller0 ~]# logout
[heat-admin@controller0 ~]$ logout
[student@workstation ~]$
```

This means that provider networks can be created using the VLAN IDs from 101 to 104 inclusive.

2. Create a VLAN provider network named `provider1-103` with a matching subnet named `subnet1-103` using the following table.

OPTION	VALUE
`--external`	
`--share`	
`--provider-network-type`	**`vlan`**
`--provider-physical-network`	**`vlanprovider1`**
`--provider-segment`	**`103`**
`--dhcp`	
`--subnet-range`	**`10.0.103.0/24`**

OPTION	VALUE
--allocation-pool	**start=10.0.103.100,end=10.0.103.149**

2.1. On `workstation` source the **/home/student/admin-rc** user environment file.

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

2.2. Create the `provider1-103` provider network.

```
[student@workstation ~]$ openstack network create \
--external \
--share \
--provider-network-type vlan \
--provider-physical-network vlanprovider1 \
--provider-segment 103 \
provider1-103
...output omitted...
| mtu                       | 1500                                 |
| name                      | provider1-103                        |
| port_security_enabled     | True                                 |
| project_id                | 42eecbfbaf684f909abfe5304434fc77     |
| provider:network_type     | vlan                                 |
| provider:physical_network | vlanprovider1                        |
| provider:segmentation_id  | 103                                  |
| qos_policy_id             | None                                 |
| revision_number           | 6                                    |
| router:external           | External                             |
...output omitted...
```

2.3. Create the `subnet1-103` subnet.

```
[student@workstation ~]$ openstack subnet create \
--dhcp \
--subnet-range=10.0.103.0/24 \
--allocation-pool=start=10.0.103.100,end=10.0.103.149 \
--network provider1-103 \
subnet1-103
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 10.0.103.100-10.0.103.149            |
| cidr              | 10.0.103.0/24                        |
| created_at        | 2018-10-25T03:19:21Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 10.0.103.1                           |
...output omitted...
```

3. Deploy an instance attached to the `provider1-103` network.

   3.1. Source the **/home/student/developer1-finance-rc** user environment file.

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

   3.2. Create an instance named `finance-server1` using the following values.

OPTION	VALUE
--flavor	**default**
--image	**rhel7**
--key-name	**example-keypair**
--network	**provider1-103**

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--network provider1-103 \
--wait \
finance-server1
...output omitted...
| addresses           | provider1-103=10.0.103.109      |
...output omitted...
```

4. From `utility`, use the **ping** command to test that `finance-server1` is reachable on the `provider1-103` network from outside of OpenStack.

```
[student@workstation ~(developer1-finance)]$ ssh utility ping -c3 10.0.103.109
PING 10.0.103.109 (10.0.103.109) 56(84) bytes of data.
64 bytes from 10.0.103.109: icmp_seq=1 ttl=64 time=1.31 ms
64 bytes from 10.0.103.109: icmp_seq=2 ttl=64 time=0.920 ms
64 bytes from 10.0.103.109: icmp_seq=3 ttl=64 time=0.545 ms

--- 10.0.103.109 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.545/0.926/1.313/0.313 ms
```

Cleanup

On `workstation`, run the **lab networking-provisioning cleanup** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab networking-provisioning cleanup
```

This concludes the guided exercise.

MANAGING OPENSTACK NETWORKING

PERFORMANCE CHECKLIST

In this lab, you will create and test VLAN provider networks.

OUTCOMES

You should be able to:

- Create a VLAN provider network.
- Launch an instance attached to a provider network.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab networking-review setup`** command. This script creates any resources required to perform the exercise.

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

1. View the network interface configuration on `utility`. `utility` has a subinterface configured in each VLAN from 101 to 104, allowing testing for VLAN traffic leaving OpenStack.
2. Using the **`/home/student/admin-rc`** environment file, create a VLAN provider network named `provider1-104` with a matching subnet named `subnet1-104`. From the previous exercise, you know that traffic for this network will travel over the `br-eth3` bridge. Use the options and values from the following table.

OPTION	VALUE
`--share`	
`--provider-network-type`	**`vlan`**
`--provider-physical-network`	**`vlanprovider1`**
`--provider-segment`	**`104`**
`--dhcp`	
`--subnet-range`	**`10.0.104.0/24`**
`--allocation-pool`	**`start=10.0.104.100,end=10.0.104.149`**

3. Using the **`/home/student/admin-rc`** environment file, create a VLAN provider network named `provider2-104` with a matching subnet named `subnet2-104`. From the previous exercise, you know that traffic for this network travels over the `br-eth4` bridge. Use the options and values from the following table.

OPTION	VALUE
`--share`	
`--provider-network-type`	`vlan`
`--provider-physical-network`	`vlanprovider2`
`--provider-segment`	`104`
`--dhcp`	
`--subnet-range`	`10.0.104.0/24`
`--allocation-pool`	`start=10.0.104.150,end=10.0.104.199`

4. Launch an instance attached to the `provider1-104` network. Name the instance `production-server1` and use the following values.

OPTION	VALUE
`--flavor`	`default`
`--image`	`rhel7`
`--key-name`	`example-keypair`
`--network`	`provider1-104`

5. Launch an instance attached to the `provider2-104` network. Name the instance `production-server2` and use the following values.

OPTION	VALUE
`--flavor`	`default`
`--image`	`rhel7`
`--key-name`	`example-keypair`
`--network`	`provider2-104`

6. From `utility`, use the **`ping`** command to test that `production-server1` and `production-server2` are reachable from outside of OpenStack.
7. Open a new terminal on `workstation`, then log in to `utility` and start the **`tcpdump`** command. This will verify that the correct VLAN is in use.
8. From `production-server1`, use the **`ping`** command to test communication with `production-server2` and `utility`.
9. On `utility`, cancel **`tcpdump`** and verify that the captured ICMP traffic is from VLAN 104.

Evaluation

On `workstation`, run the **`lab networking-review grade`** command to confirm the success of this exercise. Correct any reported failures and rerun the command until successful.

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

Cleanup

On `workstation`, run the **`lab networking-review cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab networking-review cleanup
```

This concludes the lab.

MANAGING OPENSTACK NETWORKING

PERFORMANCE CHECKLIST

In this lab, you will create and test VLAN provider networks.

OUTCOMES

You should be able to:

- Create a VLAN provider network.
- Launch an instance attached to a provider network.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab networking-review setup`** command. This script creates any resources required to perform the exercise.

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

1. View the network interface configuration on `utility`. `utility` has a subinterface configured in each VLAN from 101 to 104, allowing testing for VLAN traffic leaving OpenStack.

```
[student@workstation ~]$ ssh root@utility ip a
...output omitted...
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc fq_codel state UP group
 default qlen 1000
    link/ether 52:54:00:03:00:dc brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5054:ff:fe03:dc/64 scope link
       valid_lft forever preferred_lft forever
4: eth1.102@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UP group default qlen 1000
    link/ether 52:54:00:03:00:dc brd ff:ff:ff:ff:ff:ff
    inet 10.0.102.1/24 brd 10.0.102.255 scope global noprefixroute eth1.102
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe03:dc/64 scope link
       valid_lft forever preferred_lft forever
5: eth1.104@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UP group default qlen 1000
    link/ether 52:54:00:03:00:dc brd ff:ff:ff:ff:ff:ff
    inet 10.0.104.1/24 brd 10.0.104.255 scope global noprefixroute eth1.104
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe03:dc/64 scope link
       valid_lft forever preferred_lft forever
6: eth1.103@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UP group default qlen 1000
    link/ether 52:54:00:03:00:dc brd ff:ff:ff:ff:ff:ff
```

```
    inet 10.0.103.1/24 brd 10.0.103.255 scope global noprefixroute eth1.103
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe03:dc/64 scope link
       valid_lft forever preferred_lft forever
7: eth1.101@eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UP group default qlen 1000
    link/ether 52:54:00:03:00:dc brd ff:ff:ff:ff:ff:ff
    inet 10.0.101.1/24 brd 10.0.101.255 scope global noprefixroute eth1.101
       valid_lft forever preferred_lft forever
    inet6 fe80::5054:ff:fe03:dc/64 scope link
       valid_lft forever preferred_lft forever
```

2. Using the **/home/student/admin-rc** environment file, create a VLAN provider network named `provider1-104` with a matching subnet named `subnet1-104`. From the previous exercise, you know that traffic for this network will travel over the `br-eth3` bridge. Use the options and values from the following table.

OPTION	VALUE
--share	
--provider-network-type	**vlan**
--provider-physical-network	**vlanprovider1**
--provider-segment	**104**
--dhcp	
--subnet-range	**10.0.104.0/24**
--allocation-pool	**start=10.0.104.100,end=10.0.104.149**

2.1. On `workstation`, source the **/home/student/admin-rc** environment file.

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

2.2. Create the `provider1-104` provider network.

```
[student@workstation ~]$ openstack network create \
--share \
--provider-network-type vlan \
--provider-physical-network vlanprovider1 \
--provider-segment 104 \
provider1-104
...output omitted...
| mtu                       | 1500                                 |
| name                      | provider1-104                        |
| port_security_enabled     | True                                 |
| project_id                | 42eecbfbaf684f909abfe5304434fc77     |
| provider:network_type     | vlan                                 |
| provider:physical_network | vlanprovider1                        |
| provider:segmentation_id  | 104                                  |
| qos_policy_id             | None                                 |
| revision_number           | 6                                    |
```

```
| router:external           | External                                |
...output omitted...
```

2.3. Create the `subnet1-104` subnet.

```
[student@workstation ~]$ openstack subnet create \
--dhcp \
--subnet-range=10.0.104.0/24 \
--allocation-pool=start=10.0.104.100,end=10.0.104.149 \
--network provider1-104 \
subnet1-104
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| allocation_pools  | 10.0.104.100-10.0.104.149            |
| cidr              | 10.0.104.0/24                        |
| created_at        | 2018-10-30T03:29:46Z                 |
| description       |                                      |
| dns_nameservers   |                                      |
| enable_dhcp       | True                                 |
| gateway_ip        | 10.0.104.1                           |
...output omitted...
```

3. Using the **`/home/student/admin-rc`** environment file, create a VLAN provider network named `provider2-104` with a matching subnet named `subnet2-104`. From the previous exercise, you know that traffic for this network travels over the `br-eth4` bridge. Use the options and values from the following table.

OPTION	VALUE
`--share`	
`--provider-network-type`	**`vlan`**
`--provider-physical-network`	**`vlanprovider2`**
`--provider-segment`	**`104`**
`--dhcp`	
`--subnet-range`	**`10.0.104.0/24`**
`--allocation-pool`	**`start=10.0.104.150,end=10.0.104.199`**

3.1. Create the `provider2-104` provider network.

```
[student@workstation ~]$ openstack network create \
--share \
--provider-network-type vlan \
--provider-physical-network vlanprovider2 \
--provider-segment 104 \
provider2-104
...output omitted...
| mtu                       | 1500                                |
| name                      | provider2-104                       |
```

```
| port_security_enabled     | True                                 |
| project_id                | 42eecbfbaf684f909abfe5304434fc77     |
| provider:network_type     | vlan                                 |
| provider:physical_network | vlanprovider2                        |
| provider:segmentation_id  | 104                                  |
| qos_policy_id             | None                                 |
| revision_number           | 6                                    |
| router:external           | External                             |
...output omitted...
```

3.2. Create the `subnet2-104` subnet.

```
[student@workstation ~]$ openstack subnet create \
--dhcp \
--subnet-range=10.0.104.0/24 \
--allocation-pool=start=10.0.104.150,end=10.0.104.199 \
--network provider2-104 \
subnet2-104
+-------------------+-------------------------------------+
| Field             | Value                               |
+-------------------+-------------------------------------+
| allocation_pools  | 10.0.104.150-10.0.104.199           |
| cidr              | 10.0.104.0/24                       |
| created_at        | 2018-10-30T03:33:06Z                |
| description       |                                     |
| dns_nameservers   |                                     |
| enable_dhcp       | True                                |
| gateway_ip        | 10.0.104.1                          |
...output omitted...
```

4. Launch an instance attached to the `provider1-104` network. Name the instance `production-server1` and use the following values.

OPTION	VALUE
`--flavor`	**`default`**
`--image`	**`rhel7`**
`--key-name`	**`example-keypair`**
`--network`	**`provider1-104`**

4.1. Source the **`/home/student/operator1-production-rc`** user environment file.

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

4.2. Launch the instance.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--network provider1-104 \
```

```
--wait \
production-server1
...output omitted...
| addresses           | provider1-104=10.0.104.102      |
...output omitted...
```

5. Launch an instance attached to the `provider2-104` network. Name the instance `production-server2` and use the following values.

OPTION	VALUE
--flavor	**default**
--image	**rhel7**
--key-name	**example-keypair**
--network	**provider2-104**

5.1. Launch the instance.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--network provider2-104 \
--wait \
production-server2
...output omitted...
| addresses           | provider2-104=10.0.104.153      |
...output omitted...
```

6. From `utility`, use the **ping** command to test that `production-server1` and `production-server2` are reachable from outside of OpenStack.

```
[student@workstation ~(operator1-production)]$ ssh utility ping -c3 10.0.104.102
PING 10.0.104.102 (10.0.104.102) 56(84) bytes of data.
64 bytes from 10.0.104.102: icmp_seq=1 ttl=64 time=1.50 ms
64 bytes from 10.0.104.102: icmp_seq=2 ttl=64 time=0.797 ms
64 bytes from 10.0.104.102: icmp_seq=3 ttl=64 time=0.565 ms

--- 10.0.104.102 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2002ms
rtt min/avg/max/mdev = 0.565/0.954/1.502/0.400 ms

[student@workstation ~(operator1-production)]$ ssh utility ping -c3 10.0.104.153
PING 10.0.104.153 (10.0.104.153) 56(84) bytes of data.
64 bytes from 10.0.104.153: icmp_seq=1 ttl=64 time=1.23 ms
64 bytes from 10.0.104.153: icmp_seq=2 ttl=64 time=0.625 ms
64 bytes from 10.0.104.153: icmp_seq=3 ttl=64 time=0.565 ms

--- 10.0.104.153 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2001ms
rtt min/avg/max/mdev = 0.565/0.809/1.238/0.304 ms
```

7. Open a new terminal on `workstation`, then log in to `utility` and start the **`tcpdump`** command. This will verify that the correct VLAN is in use.

```
[student@workstation ~]$ ssh root@utility
[root@utility ~]# tcpdump -nnei eth1 -vvv
```

8. From `production-server1`, use the **`ping`** command to test communication with `production-server2` and `utility`.

   8.1. From `workstation`, obtain 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=84f2c5ae-           |
|       | 11b9-4297-be63-327fcc7152f4                                       |
+-------+-------------------------------------------------------------------+
```

   8.2. Open the link in Firefox, then log in as `root` with the password `redhat`.

   8.3. Use the **`ping`** command to test the connection to `production-server2` and `utility`.

```
[root@production-server1 ~]# ping -c3 10.0.104.153
PING 10.0.104.153 (10.0.104.153) 56(84) bytes of data.
64 bytes from 10.0.104.153: icmp_seq=1 ttl=64 time=3.23 ms
64 bytes from 10.0.104.153: icmp_seq=2 ttl=64 time=1.22 ms
64 bytes from 10.0.104.153: icmp_seq=3 ttl=64 time=0.863 ms

--- 10.0.104.153 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.863/1.774/3.234/1.043 ms
[root@production-server1 ~]# ping -c3 10.0.104.1
PING 10.0.104.1 (10.0.104.1) 56(84) bytes of data.
64 bytes from 10.0.104.1: icmp_seq=1 ttl=64 time=3.23 ms
64 bytes from 10.0.104.1: icmp_seq=2 ttl=64 time=1.22 ms
64 bytes from 10.0.104.1: icmp_seq=3 ttl=64 time=0.863 ms

--- 10.0.104.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 2003ms
rtt min/avg/max/mdev = 0.863/1.774/3.234/1.043 ms
```

9. On `utility`, cancel **`tcpdump`** and verify that the captured ICMP traffic is from VLAN 104.

```
...output omitted...
05:41:39.550482 fa:16:3e:b7:86:40 > 52:54:00:03:00:dc, ethertype 802.1Q (0x8100),
 length 102: vlan 104, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 11745, offset 0,
 flags [DF], proto ICMP (1), length 84)
    10.0.104.102 > 10.0.104.1: ICMP echo request, id 1339, seq 1, length 64
05:41:39.550564 52:54:00:03:00:dc > fa:16:3e:b7:86:40, ethertype 802.1Q (0x8100),
 length 102: vlan 104, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 46758, offset 0,
 flags [none], proto ICMP (1), length 84)
    10.0.104.1 > 10.0.104.102: ICMP echo reply, id 1339, seq 1, length 64
```

```
05:41:40.552250 fa:16:3e:b7:86:40 > 52:54:00:03:00:dc, ethertype 802.1Q (0x8100),
 length 102: vlan 104, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 12450, offset 0,
 flags [DF], proto ICMP (1), length 84)
    10.0.104.102 > 10.0.104.1: ICMP echo request, id 1339, seq 2, length 64
05:41:40.552308 52:54:00:03:00:dc > fa:16:3e:b7:86:40, ethertype 802.1Q (0x8100),
 length 102: vlan 104, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 47597, offset 0,
 flags [none], proto ICMP (1), length 84)
    10.0.104.1 > 10.0.104.102: ICMP echo reply, id 1339, seq 2, length 64
05:41:41.553772 fa:16:3e:b7:86:40 > 52:54:00:03:00:dc, ethertype 802.1Q (0x8100),
 length 102: vlan 104, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 12729, offset 0,
 flags [DF], proto ICMP (1), length 84)
    10.0.104.102 > 10.0.104.1: ICMP echo request, id 1339, seq 3, length 64
05:41:41.553837 52:54:00:03:00:dc > fa:16:3e:b7:86:40, ethertype 802.1Q (0x8100),
 length 102: vlan 104, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 48352, offset 0,
 flags [none], proto ICMP (1), length 84)
    10.0.104.1 > 10.0.104.102: ICMP echo reply, id 1339, seq 3, length 64
...output omitted...
```

Evaluation

On `workstation`, run the **`lab networking-review grade`** command to confirm the success of this exercise. Correct any reported failures and rerun the command until successful.

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

Cleanup

On `workstation`, run the **`lab networking-review cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab networking-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- 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.
- Open Virtual Networking (OVN) is the SDN networking project that extends OVS to provide layer 2 and layer 3 network functions. It utilizes the Geneve framework to create tunnels between OpenStack nodes.
- The Modular Layer 2 (ML2) plug-in is a framework that enables the use of various technologies. Administrators can interact with Open vSwitch or any vendor technology, such as Cisco equipment, using the various plug-ins available for OpenStack networking.
- Self-service and provider networks can be used in combination or exclusively to define the network connection types available to instances. Provider networks give instances a layer 2 connection to networks outside of OpenStack, whereas tenant networks require an OpenStack router to connect them to an external network.

CHAPTER 7

MANAGING COMPUTE RESOURCES

GOAL

Describe the processes and architecture of both regular and hyperconverged compute nodes. Perform common administration tasks for compute nodes.

OBJECTIVES

- Explain the end-to-end launch process for server instances, including scheduler and conductor tasks.
- Describe the hyperconverged compute node architecture and manage scheduling methods for sets of compute nodes.
- Perform common compute node administration tasks, including live migration, evacuation, and enabling and disabling compute nodes.

SECTIONS

- Describing the Instance Launch Process (and Guided Exercise)
- Introducing Red Hat Hyperconverged Infrastructure (and Guided Exercise)
- Administering Compute Nodes (and Guided Exercise)

LAB

- Managing Compute Resources

DESCRIBING THE INSTANCE LAUNCH PROCESS

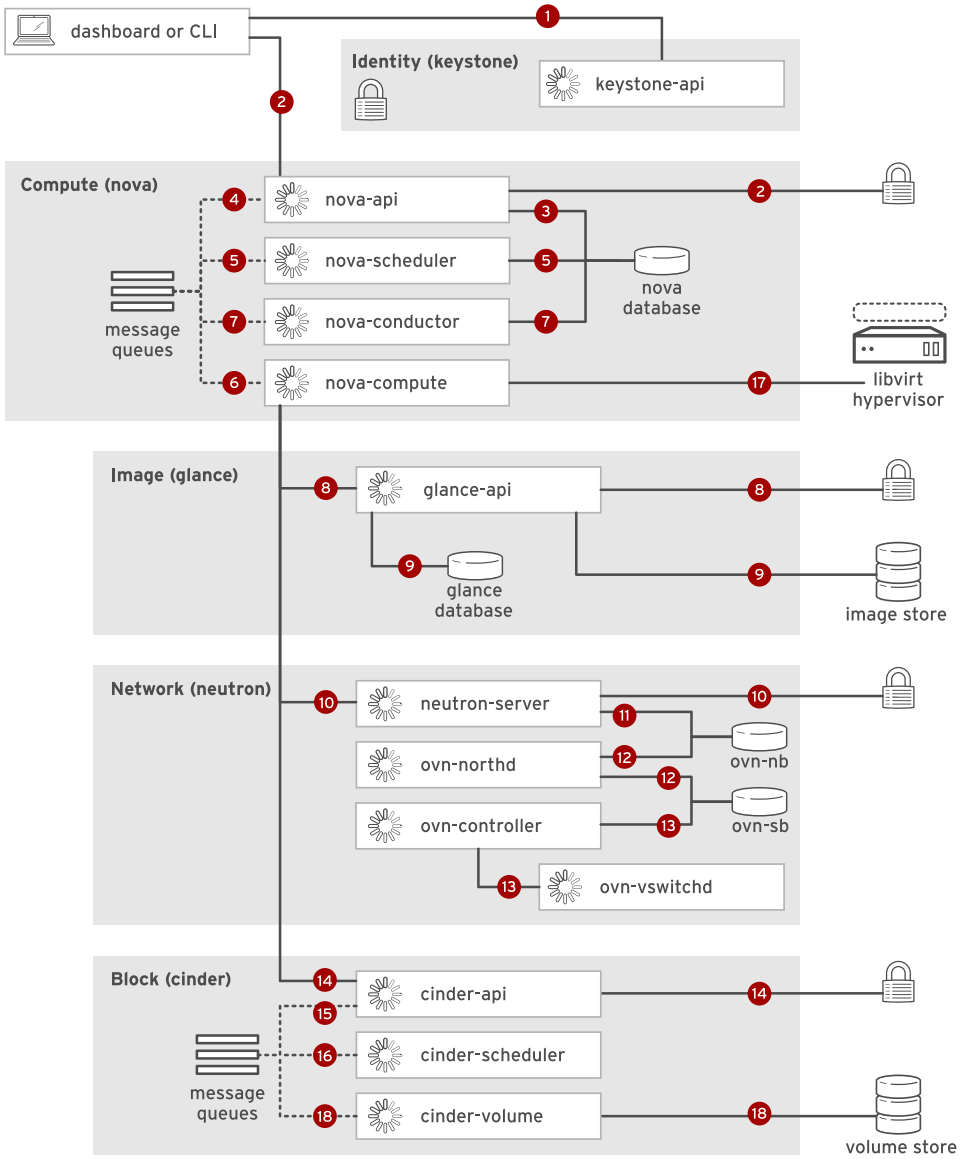
OBJECTIVES

After completing this section, students should be able to:

- Explain the launch process.
- Discuss the compute scheduling and compute placement services.

INSTANCE LAUNCH PROCESS

The launch process uses a combination of REST API calls to OpenStack services, and message broker RPC calls for coordination between service components.



Instance Launch Process

1. Using the dashboard or the command-line client, the user's account credentials are sent as a REST API call to the `identity` service endpoint. After successful authentication, the `identity` service generates and returns an *authentication token (auth token)* to the user. The auth token is included with subsequent REST calls to other services, to permit those services to perform actions on behalf of this user.

2. The instance launch request and the auth token are sent as a REST API call to the `compute` service endpoint. `nova_api` relays the request to the `identity` service, which validates the auth token and returns allowed roles and permissions in the updated token header.

3. `nova_api` creates a persistent database entry for the new instance, including the launch parameters in the original request. Other processes can obtain data from this entry, and update the current state to this persistent entry.

4. `nova_api` publishes an rpc.call to a named message queue, requesting `nova_scheduler` to locate an acceptable compute node to host the new instance.

5. `nova_scheduler` subscribes to the new instance request, reads filtering and weighing parameters from the instance database entry, reads cluster compute node data from the database, and updates the instance record with the selected compute host ID. `nova_scheduler` submits an rpc.call to a message queue, requesting that `nova_compute` initiate an instance launch.

6. `nova_compute` subscribes to the new instance request, then publishes an rpc.call for `nova_conductor` to prepare for an instance launch.

7. `nova_conductor` subscribes to the new instance request, reads the database entry to obtain the compute host ID and instance flavor with requested RAM, vCPUs, and disk configuration, and then publishes the new instance state to the message queue.

8. `nova_compute` subscribes to the new instance request to retrieve the instance information. Using the image ID from the instance request, `nova_compute` sends a REST API call to the `image service` to obtain the requested image's URL. The `glance_api` service forwards the request to the `identity` service, which again validates the auth token and returns allowed roles and permissions in the updated token header.

9. Using the image ID, `glance_api` retrieves image metadata from the `image` service database, and returns the image URL to `nova_compute`. `nova_compute` loads the image from the image store using the URL.

10. `nova_compute` sends a REST API call to the `networking` service, requesting allocation and configuration of network resources for the new instance. `neutron_server` forwards the request to the `identity` service, which again validates the auth token and returns allowed roles and permissions in the updated token header.

11. `neutron_server`, as a cloud management system plug-in, reads and writes logical network resource definitions to the `ovn-nb` database. `neutron_server` checks for the existing network and creates new resources for needed ports, connections, and network parameters.

12. The `ovn-northd` daemon reads from `ovn-nb` and translates the logical network configuration into logical data-path flows in the `ovn-sb` database. The additional physical network and port bindings are populated by the `ovn-controller` on the selected compute host.

13. The `ovn-controller` reads the configuration from `ovn-sb` and updates the compute host's status in the physical network and the binding tables. `ovn-controller` connects to `ovs-vswitchd` as an OpenFlow controller to dynamically configure control over network traffic

through OpenFlow rules on OVS bridges. `neutron_server` returns the L2 configuration obtained from the compute host libvirt driver, and the requested DHCP address, to the message queue. `nova_compute` writes this instance network state to the instance database entry.

14. `nova_compute` sends a REST API call to the block storage service, requesting volume data for the new instance disks. `cinder_api` forwards the request to the identity service, which again validates the auth token and returns allowed roles and permissions in the updated token header.

15. `cinder_api` publishes an rpc.call to a named message queue, requesting that `cinder_scheduler` create a volume of the requested size, or to locate the volume metadata for a volume that already exists.

16. `cinder_scheduler` subscribes to the new instance request, creates or locates the volume, and then returns the requested metadata to `nova_scheduler`.

17. `nova_compute` generates data for the compute node driver and executes the request using libvirt to create a VM on the compute host. The VM appears in the dashboard and in commands that list servers. `nova_compute` passes the volume information to `libvirt`.

18. `cinder_volume` subscribes to the new instance request, and retrieves the cluster map. `Libvirt` then mounts the volume on the new instance.

Through the `identity` service, the user can query the instance state by a REST API call to `nova_api`, which then retrieves the current state from the compute service database.

SCHEDULING

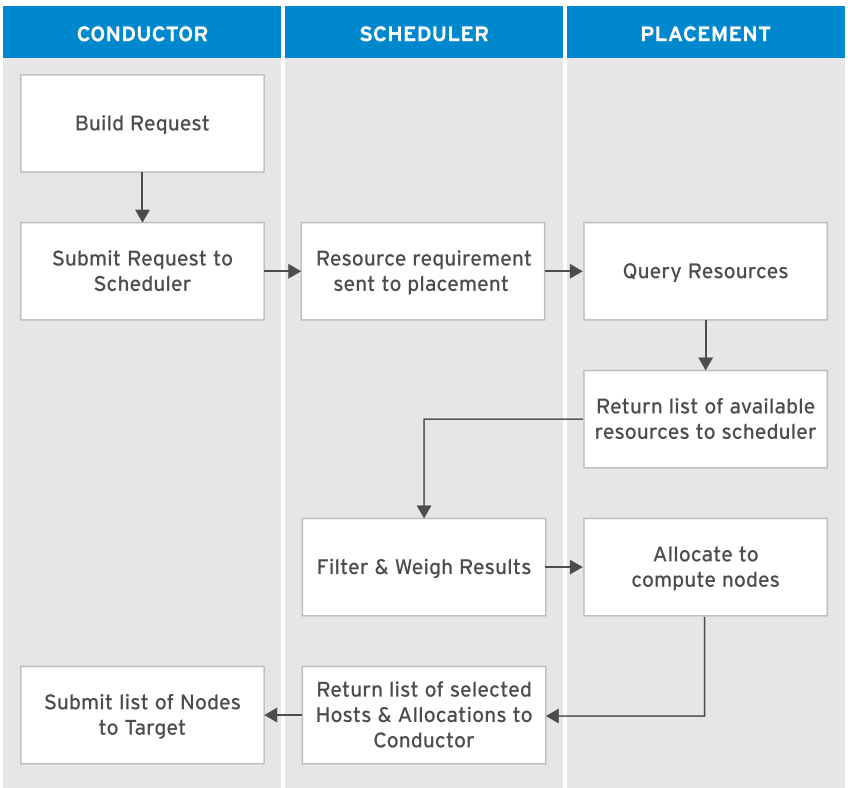


Figure 7.1: Scheduling process

The `scheduler` receives a request from the `conductor`. The request contains the resource requirements which are sent to the `placement` service. The `placement` service runs a query which returns all available compute nodes able to satisfy the resource requirements. The `placement` service creates a summary for each compute node that matches the resource requirements. The summary is sent to the `scheduler`.

The `scheduler` runs the requirements through filters to refine the search. The compute nodes are ranked and the compute node at the top of the list is chosen. The scheduler then sends the request to the placement service to claim the resources. The claim can return an unsuccessful result if, in the meantime, the resources have been claimed by another request. In this case, the scheduler selects the next compute node in the ranking list. If the claim is successful the scheduler contacts the conductor. The conductor attempts to build the instance on the selected host. If the build fails for any reason, the conductor ensures that the resources are not claimed by the instance.

PLACEMENT SERVICE

The `placement service` was introduced in the Red Hat OpenShift Platform 10 release of OpenStack. Before this release, the placement service tasks were carried out by the `compute` service. The data model is used to find resources, for example, compute nodes, storage pools, or an IP allocation pool. The resources of each provider are tracked by the `placement` service. Resources on compute nodes are RAM and CPU. Resources on storage nodes are disks. Consumed resources are tracked as classes.

Generic Resource Pools

In the cloud environment there are various kinds of resources. Resources provided by the compute nodes include RAM, CPU, PCI devices, and ephemeral disks. Other resources, for example the shared storage, are provided by an external resource pool. Previously, the `compute` service considered all resources as being provided by a compute node. This caused issues with resource tracking, meaning that usage and capacity values were calculated inaccurately.

A new RESTful API has been implemented to address the shared storage resource problem. The API allows the placement service to query and manage resource providers, inventories, and allocation records. The resource providers provide consumption information to all consumers of the resources. These generic resource pools solve the problem of incorrect capacity and usage information. The API allows administrative service users to create resource providers. Capacity and usage information is updated for all resource providers, thus indicating which compute nodes can use a specific resource by associating them to one or more resource provider aggregates.

For example, an administrator can create a resource provider aggregate comprising all Ceph storage nodes in a specific rack. A resource pool is then created to represent the Ceph storage. The **`resource-provider`** command makes two REST API requests: the first to create the resource provider, and the second to associate the resource provider aggregate. The resource pool capacity is then updated with the total amount of shared disk space.

Qualitative Resources or Traits

The resource provider manages *quantitative* aspects at the boot request. It has a collection of *inventory* and *allocation* objects to manage these quantitative requests. The inventories are subtracted from the allocations when an instance uses resources from a resource provider. However, the resource provider also needs to manage non-consumable, or *qualitative*, resources.

For example, a user may request 80 GB of disk space for an instance. This resource is quantitative. However, the user may also request an SSD disk. This resource is qualitative.

A new REST resource in the placement API manages qualitative information. There are standard traits and custom traits. Standard traits can be used across different cloud environments. Standard traits cannot be modified.

Custom traits are used by administrative users to define nonstandard qualitative information. This information is used by the resource providers. By managing the characteristics of the resource providers the placement service can help the scheduler make better placement decisions. The traits API is used to store and query the qualitative resources.

For example, an administrator can add a trait to an existing resource provider aggregate, tagging it as SSD storage. This standard trait is then read by the compute node resource provider. When a user creates a new instance, specifying that it must use an SSD drive, the scheduler sends that information to the placement service. The placement service uses the aggregate to return the relevant information to the scheduler.

In previous versions of OpenStack, resource provider aggregate metadata would have been used to manage this qualitative data. However, this practice is hard to manage and is not scalable. Further, aggregate metadata only works for compute node resources.

Eventually the use of traits will deprecate resource provider aggregate metadata, but aggregates will remain as they control much more than just compute node metadata.

REFERENCES

Placement API

https://docs.openstack.org/nova/rocky/user/placement.html

DESCRIBING THE INSTANCE LAUNCH PROCESS

In this exercise, you will view the placement server's log files and data and observe scheduler messages.

OUTCOMES

You should be able to view the placement service log file, and observe scheduler messages for both a successful and an unsuccessful instance creation.

BEFORE YOU BEGIN

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

Run the **`lab computeresources-launch setup`** command. This script ensures that OpenStack services are running and the environment is properly configured for the guided exercise.

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

1. From `workstation`, log on to the `controller0` node and change to the `root` user. Configure the compute services for debugging, then restart the containers and confirm their status.

   1.1. From `workstation`, log on to `controller0` as the `heat-admin` user and change to the `root` user.

```
[student@workstation ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

   1.2. Edit the **`/var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf`** file and set `debug` to **`true`**. Restart the `nova_scheduler` and `nova_api` containers, then wait for 45 seconds. Use the **`docker ps`** command to ensure the containers are running and healthy.

```
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf \
DEFAULT debug True
[root@controller0 ~]# docker restart nova_api
nova_api
[root@controller0 ~]# docker restart nova_scheduler
nova_scheduler
[root@controller0 ~]# docker ps \
--format="{{.Names}}\t{{.Status}}" | grep nova
...output omitted...
nova_api        Up About 31 seconds (healthy)
nova_scheduler  Up About 46 seconds (healthy)
```

```
...output omitted...
```

2. Open three additional terminal windows. From `workstation`, log in to the `compute0` node. From `workstation`, log in to the `controller0` node. From `workstation`, log in to the `controller0` node. Use the **`sudo -i`** command to gain root privileges on each machine. Change into the **`/var/log/containers/nova`** directory on each node.

   These extra terminal windows will be used to inspect the compute service log files.

   2.1. Open a new terminal window and log in to the `compute0` node as the `heat-admin` user. Use the **`sudo -i`** command to gain root privileges. Change into the log directory.

```
[student@workstation ~]$ ssh heat-admin@compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]# cd /var/log/containers/nova
[root@compute0 nova]#
```

   2.2. Open a new terminal window and log in to the `controller0` node as the `heat-admin` user. Use the **`sudo -i`** command to gain root privileges. Change into the log directory.

```
[student@workstation ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# cd /var/log/containers/nova
[root@controller0 nova]#
```

   2.3. Open a new terminal window and log in to the `controller0` node as the `heat-admin` user. Use the **`sudo -i`** command to gain root privileges. Change into the log directory.

```
[student@workstation ~]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# cd /var/log/containers/nova
[root@controller0 nova]#
```

3. On `workstation`, as the `architect1` user, create a new instance called `finance-server1`.

   3.1. Source the **`/home/student/architect1-finance-rc`** environment identity file.

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

   3.2. Use the **`openstack server create`** command to create an instance called `finance-server1`. Use the `rhel7` image, the `default` flavor, the `default` security group, and the `finance-network1` network. Specify `compute0` as the availability zone. Note the ID of the instance, because it will be used to inspect the nova log files.

```
[student@workstation ~(architect1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--nic net-id=finance-network1 \
--security-group finance-web \
```

```
--availability-zone nova:compute0.overcloud.example.com \
--wait finance-server1
+-------------------------------------+-------------------------------------+
| Field                               | Value                               |
+-------------------------------------+-------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                              |
| OS-EXT-AZ:availability_zone         | nova                                |
| OS-EXT-SRV-ATTR:host                | compute0.overcloud.example.com      |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute0.overcloud.example.com      |
...output omitted...
| id                                  | 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b |
...output omitted...
```

4. On `compute0`, inspect the **`nova-compute.log`** log file. Use the **`grep`** command to list the entries that relate to the ID of `finance-server1`.

```
[root@compute0 nova]# grep 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b nova-compute.log
```

There are several entries in the **`nova-compute.log`** file for the creation of `finance-server1`. Notice the entries where the compute service is attempting to claim space on `compute0`. Also notice that this log entry informs the user about the memory, disk, and vCPUs required by `finance-server1`.

```
2018-11-01 09:34:08.050 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Attempting claim on node
 compute0.overcloud.example.com: memory 2048 MB, disk 10 GB, vcpus 2 CPU
```

The next entries inform the user about the memory, disk, and vCPUs available on `compute0`.

```
2018-11-01 09:34:08.050 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Total memory: 6143 MB, used:
 4096.00 MB
2018-11-01 09:34:08.050 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] memory limit not specified,
 defaulting to unlimited
2018-11-01 09:34:08.050 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Total disk: 113 GB, used: 0.00
 GB
2018-11-01 09:34:08.051 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
```

```
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] disk limit not specified,
 defaulting to unlimited
2018-11-01 09:34:08.051 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Total vcpu: 4 VCPU, used: 0.00
 VCPU
2018-11-01 09:34:08.051 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] vcpu limit not specified,
 defaulting to unlimited
```

The following entry informs the user that the claim by the nova compute service was successful.

```
2018-11-01 09:34:08.051 1 INFO nova.compute.claims
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Claim successful on node
 compute0.overcloud.example.com
```

The remaining entries inform the user that the instance is being created. The final entry tells the user how long it took to build the instance.

```
2018-11-01 09:34:09.021 1 INFO nova.virt.libvirt.driver
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Creating image
2018-11-01 09:34:29.110 1 INFO nova.compute.manager [req-92997d21-381c-4cba-9d97-
c83d5d51cee8 d5ea361ac1d44b3fa6aa289cd0a52756 d8f51de41e7e4f4b8c71ed7dfc226c97
 - default default] [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] VM Started
 (Lifecycle Event)
...output omitted...
2018-11-01 09:34:34.016 1 INFO nova.compute.manager [req-bd300177-3613-4ec6-8bd7-
da20a7857a6b d5ea361ac1d44b3fa6aa289cd0a52756 d8f51de41e7e4f4b8c71ed7dfc226c97
 - default default] [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] Took 25.98
 seconds to build instance.
```

Notice that in all the log entries, before the *Life-cycle Event* starts, all contain the same ID. It appears just before the ID for `finance-server1`. This ID will be used in the next steps to inspect the **`nova-scheduler.log`** log file.

```
2018-11-01 09:34:08.051 1 INFO nova.compute.claims [req-
c41919ad-a82e-43b0-89eai-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default]
 [instance: 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b] vcpu limit not specified,
 defaulting to unlimited
```

5. On one of the `controller0` terminals, use the **`grep`** command and the ID of `finance-server1` to list the entries in the **`nova-placement`** log file. There are three entries created for each instance created. Notice that there are two *GET* entries and one *PUT* entry.

```
[root@controller0 nova]# grep 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b \
nova-placement.log
2018-11-01 09:34:07.772 12 INFO nova.api.openstack.placement.requestlog
 [req-7980d973-6aac-49bf-b58c-d44e1f90a084 a48dc9ff61c54efb8104440971e1ffdd
 d8f51de41e7e4f4b8c71ed7dfc226c97 - default default] 172.24.1.1 "GET /
placement/allocations/1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b" status: 200 len: 19
 microversion: 1.0
2018-11-01 09:34:07.811 12 INFO nova.api.openstack.placement.requestlog
 [req-cbbb796e-5d99-4e04-a2c0-ed62ec389cd9 a48dc9ff61c54efb8104440971e1ffdd
 d8f51de41e7e4f4b8c71ed7dfc226c97 - default default] 172.24.1.1 "PUT /placement/
allocations/1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b" status: 204 len: 0 microversion:
 1.17
2018-11-01 09:34:08.956 12 INFO nova.api.openstack.placement.requestlog
 [req-c7076539-5929-457c-bbde-8069e990ea9b a48dc9ff61c54efb8104440971e1ffdd
 d8f51de41e7e4f4b8c71ed7dfc226c97 - default default] 172.24.1.1 "GET /placement/
allocations/1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b" status: 200 len: 136
 microversion: 1.0
```

6. On one of the `controller0` terminals, use the **`grep`** command with the ID copied from the **`nova-compute.log`** file to list the entries in the **`nova-scheduler`** log file.

```
[root@controller0 nova]# grep 8dd2d316acc74f54b890bae519ea75b9 nova-scheduler.log
```

The first entries are a series of host *locking* and *releasing* actions by the `nova_scheduler` service.

```
2018-11-01 09:33:54.461 1 DEBUG oslo_concurrency.lockutils
 [req-58612dbf-305f-4a13-b1e1-638a65324c0a
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default] Lock
 "host_instance" acquired by "nova.scheduler.host_manager.delete_instance_info" ::
 waited 0.000s inner /usr/lib/python2.7/site-packages/oslo_concurrency/
lockutils.py:273
        2018-11-01 09:33:54.461 1 DEBUG oslo_concurrency.lockutils
 [req-58612dbf-305f-4a13-b1e1-638a65324c0a
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default] Lock
 "host_instance" released by "nova.scheduler.host_manager.delete_instance_info" ::
 held 0.000s inner /usr/lib/python2.7/site-packages/oslo_concurrency/
lockutils.py:285
```

In this extract, the first entry shows `nova_scheduler` starting the schedule for instance `finance-server1`. The next one shows `nova_scheduler` getting the compute nodes and services for instance creation.

```
2018-11-01 09:34:06.397 1 DEBUG nova.scheduler.manager
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
```

```
 default] Starting to schedule for instances:
 [u'1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b'] select_destinations /usr/lib/python2.7/
site-packages/nova/scheduler/manager.py:110
        2018-11-01 09:34:06.934 1 DEBUG nova.scheduler.host_manager
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Getting compute nodes and services for cell
 00000000-0000-0000-0000-000000000000(cell0) _get_computes_for_cells /usr/lib/
python2.7/site-packages/nova/scheduler/host_manager.py:623
```

The `nova_scheduler` service then checks for host aggregates.

```
2018-11-01 09:34:06.958 1 DEBUG nova.scheduler.host_manager
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Update host state with aggregates: [] _locked_update /usr/lib/python2.7/
site-packages/nova/scheduler/host_manager.py:175
```

7. On the same `controller0` node, and using the same **`grep`** command, search for the ID of `finance-server1`.

```
[root@controller0 nova]# grep 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b nova-
scheduler.log
```

There are three entries for the creation of each instance. The first entry shows `nova_scheduler` scheduling the instance. The second entry shows `nova_scheduler` claiming resources. The third entry is the creation of the instance.

```
2018-11-01 09:34:06.397 1 DEBUG nova.scheduler.manager
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Starting to schedule for instances:
 [u'1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b'] select_destinations /usr/lib/python2.7/
site-packages/nova/scheduler/manager.py:110
2018-11-01 09:34:06.961 1 DEBUG nova.scheduler.utils
 [req-c41919ad-a82e-43b0-89ea-6b268a64d15b
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Attempting to claim resources in the placement API for instance
 1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b claim_resources /usr/lib/python2.7/site-
packages/nova/scheduler/utils.py:786
2018-11-01 10:10:29.698 1 DEBUG nova.scheduler.host_manager
 [req-a56f6380-23a3-4ba4-9296-69075401800d
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Update host state with instances:
 {u'1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b':
 Instance(access_ip_v4=None,access_ip_v6=None,architecture=None,
 auto_disk_config=False,availability_zone='nova',cell_name=None,
 cleaned=False,config_drive='',created_at=2018-11-01T09:34:06Z,
```

```
 default_ephemeral_device=None,default_swap_device=None,
 deleted=False,deleted_at=None,device_metadata=None,
disable_terminate=False,display_description='finance-
server1',display_name='finance-server1',
ec2_ids=EC2Ids,ephemeral_gb=0,ephemeral_key_uuid=None,fault=
...output omitted...
flavor=Flavor(2),host='compute0.overcloud.example.com',hostname='finance-
server1',id=4,image_ref='6b0128a9-4481-4ceb-b34e-
ffe92e0dcfdd',info_cache=InstanceInfoCache,instance_type_id=2,kernel_id='',
 key_data=None,key_name=None,keypairs=KeyPairList,launch_index=0,
  launched_at=2018-11-01T09:34:33Z,launched_on='compute0.overcloud.example.com',
 locked=False,locked_by=None,memory_mb=2048,metadata={},migration_context=
...output omitted...
image_container_format='bare',image_cpu_arch='x86_64',image_disk_format='qcow2',
image_min_disk='10',image_min_ram='2048',image_owner_specified.shade.md5=
'9300685c8287e30b967ca4cb71057b3b',image_owner_specified.shade.object='images/
rhel7',image_owner_specified.shade.sha256=
 'e5855605fdce3754c3d600f6d3b8a9bd58e93371b376fc7c6b5af8dc9c16d952',
owner_project_name='finance',owner_user_name='architect1'},
tags=TagList,task_state=None,terminated_at=None,updated_at=2018-11-01T09:34:09Z,
user_data=None,user_id=
'69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726',
uuid=1dd346b7-8b39-4b0f-988b-90f1c3d4ef3b,vcpu_model=VirtCPUModel,
vcpus=2,vm_mode=None,vm_state='active')} _locked_update /usr/lib/python2.7/site-
packages/nova/scheduler/host_manager.py:181
```

8. Use the **`openstack server create`** command to create an instance called `finance-server2`. Use the `rhel7` image, the `launch` flavor, the `finance-web` security group, and the `finance-network1` network. Specify `compute0` as the availability zone.

```
[student@workstation ~(architect1-finance)]$ openstack server create \
--image rhel7 \
--flavor launch \
--nic net-id=finance-network1 \
--security-group finance-web \
--availability-zone nova:compute0.overcloud.example.com \
--wait finance-server2
```

The instance launch will fail because the flavor's memory is too small for the image.

```
Flavor's memory is too small for requested image. (HTTP 400) (Request-
ID: req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513)
```

9. On a `controller0` node, inspect the **`nova-api.log`** log file. Use the *request ID* from the error message received following the failed instance creation.

```
[root@controller0 nova]# grep req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513 \
nova-api.log
```

The first entry shows the attempt to create the server on `compute0`.

```
2018-11-01 10:52:46.193 14 DEBUG nova.api.openstack.wsgi
 [req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513
```

```
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Action: 'create', calling method: <bound method ServersController.create
 of <nova.api.openstack.compute.servers.ServersController object
 at 0x7f42cb10fd50>>, body: {"server": {"name": "finance-server2",
 "imageRef": "6b0128a9-4481-4ceb-b34e-ffe92e0dcfdd", "availability_zone":
 "nova:compute0.overcloud.example.com", "flavorRef": "10b69784-0ed5-4394-982c-
b6239e663a9c", "max_count": 1, "min_count": 1, "networks": [{"uuid":
 "7a6556ab-6083-403e-acfc-79caf3873660"}]}} _process_stack /usr/lib/python2.7/
site-packages/nova/api/openstack/wsgi.py:604
```

The next entry shows the lock obtained by the `nova_scheduler` service.

```
2018-11-01 10:52:46.214 14 DEBUG oslo_concurrency.lockutils
 [req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Lock "00000000-0000-0000-0000-000000000000" acquired by
 "nova.context.get_or_set_cached_cell_and_set_connections" :: waited 0.000s
 inner /usr/lib/python2.7/site-packages/oslo_concurrency/lockutils.py:273
```

Some entries later the `nova_scheduler` service obtains quotas for the project.

```
2018-11-01 10:52:46.740 14 DEBUG nova.quota
 [req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Getting quotas for project 3c003f65d8d64914a053f178fbbf953c. Resources:
 ['injected_files'] _get_quotas /usr/lib/python2.7/site-packages/nova/quota.py:409
```

Some entries later notice the error message informing the user that the flavor's memory is too small.

```
2018-11-01 10:52:46.762 14 INFO nova.api.openstack.wsgi
 [req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9 default] HTTP
 exception thrown: Flavor's memory is too small for requested image.
        2018-11-01 10:52:46.763 14 DEBUG nova.api.openstack.wsgi
 [req-5f1ca5b7-94ee-4a62-9703-3cbe78e6c513
 69fb452af3dc1c1b54fb342df19d898fe3928e50cc930ebb8f112b1a59e91726
 3c003f65d8d64914a053f178fbbf953c - 8dd2d316acc74f54b890bae519ea75b9
 default] Returning 400 to user: Flavor's memory is too small for requested image.
 __call__ /usr/lib/python2.7/site-packages/nova/api/openstack/wsgi.py:1064
```

Close all the terminal windows used in this guided exercise.

Cleanup

On `workstation`, run the **`lab computeresources-launch cleanup`** script to clean up the resources created in this exercise.

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

This concludes the guided exercise.

INTRODUCING RED HAT HYPERCONVERGED INFRASTRUCTURE

OBJECTIVES

After completing this section, students should be able to:

- Discuss *Red Hat Hyperconverged Infrastructure (HCI)* node architecture.
- Create host aggregates.

INTRODUCTION TO HCI NODES

An overcloud is deployed using roles that determine the services and components installed on each node. The predefined roles in our typical classroom deployment include `Controller`, `Compute`, and `CephStorage`, but numerous other predefined roles exist. This section introduces another predefined role, called *ComputeHCI*. It is used to deploy a *hyperconverged* compute node. Hyperconverged is a node configuration that combines a normal hypervisor compute role with local Ceph OSDs on the same compute node. The Red Hat Hyperconverged Infrastructure always uses Ceph as the storage component. The Ceph components on a hyperconverged node are identical to the components installed on a Ceph Storage role storage node.

> **NOTE**
> Red Hat offers two related but separate hyperconverged solutions. The first, called Red Hat Hyperconverged Infrastructure for Virtualization, combines Red Hat Gluster Storage with Red Hat Virtualization (RHV) in an RHV hypervisor cluster. The second is the Red Hat OpenStack Platform solution introduced here, combining RHOSP 13 and Red Hat Ceph Storage 3, and is called Red Hat Hyperconverged Infrastructure for Cloud.

The benefit of hyperconverged storage is lower cost and greater operational flexibility through more efficient resource management. Remote branches and edge networks are a primary use case scenario for hyperconverged nodes.

There are two methods for including hyperconverged nodes in an overcloud deployment. The first does not require configuring or assigning the ComputeHCI role. Instead it sets a parameter that declares that the Compute role nodes should be configured as the ComputeHCI role instead. Using this method, all compute nodes will become hyperconverged nodes, and no normal compute nodes will be created. The second method is to enable the ComputeHCI role, and then assign either the Compute or the ComputeHCI role to each intended compute node. The latter method is the only way to deploy a mix of compute node types in a single RHOSP cluster.

Overcloud nodes are deployed using RHOSP Director with deployment configuration templates. View the configured roles in the role template file that was used to deploy this classroom overcloud.

```
[stack@director]$ cd /home/stack/templates
[stack@director]$ grep ' name' roles_data.yaml
- name: Controller
- name: Compute
#- name: BlockStorage
#- name: ObjectStorage
```

```
- name: CephStorage
- name: ComputeHCI
```

View the network port specification that determines which OpenStack management networks will have an interface on nodes assigned with this role, found in the network configuration template file.

```
[stack@director]$ cd /home/stack/templates/classroom-environment
[stack@director]$ grep ComputeHCI:: 34-ips-from-pool-all.yaml
    OS::TripleO::ComputeHCI::Ports::ExternalPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/external_from_pool.yaml
    OS::TripleO::ComputeHCI::Ports::InternalApiPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/internal_api_from_pool.yaml
    OS::TripleO::ComputeHCI::Ports::StoragePort: /usr/share/openstack-tripleo-
heat-templates/network/ports/storage_from_pool.yaml
    OS::TripleO::ComputeHCI::Ports::StorageMgmtPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/storage_mgmt_from_pool.yaml
    OS::TripleO::ComputeHCI::Ports::TenantPort: /usr/share/openstack-tripleo-heat-
templates/network/ports/tenant_from_pool.yaml
    OS::TripleO::ComputeHCI::Ports::ManagementPort: /usr/share/openstack-tripleo-
heat-templates/network/ports/management_from_pool.yaml
```

The IP addresses assigned to these network interfaces are in the same network configuration template file. You can compare this information to the interfaces on the *computehci0* node.

```
[stack@director]$ cd /home/stack/templates/classroom-environment
[stack@director]$ grep ComputeHCIIPs: -A 14 34-ips-from-pool-all.yaml
ComputeHCIIPs:
external:
- 172.25.250.6
internal_api:
- 172.24.1.6
tenant:
- 172.24.2.6
storage:
- 172.24.3.6
storage_mgmt:
- 172.24.4.6
management:
- 172.24.5.6
```

A hyperconverged infrastructure offers cost and space savings by allowing the use of standardized private cloud building blocks, making it easier to manage compute and storage functions as a single unit. Using hyperconverged infrastructure improves application portability between the data center and remote networks, and provides greater leverage for network function virtualization deployments at the network edge.

In RHOSP 13, all overcloud services are containerized, which adds resource management efficiency and increased cost savings to each hyperconverged node implementation. A hyperconverged infrastructure also allows organizations to replace siloed specialist storage roles with OpenStack cloud administrators.

Red Hat OpenStack Platform allows for a nonuniform mix of standard compute nodes, Ceph storage nodes, and hyperconverged nodes. The RHOSP infrastructure can scale to match evolving

growth and application requirements patterns in any proportion of compute and storage resources, in both centralized and distributed node configurations.

HYPERCONVERGED CONFIGURATION

To create an HCI environment, ensure that the following configuration is completed before the overcloud is deployed.

Resource Isolation

Hyperconverged nodes combine the compute service and Ceph storage OSD services on the same system. By default, the compute service is not aware that system resources are being shared with another significant service. Hyperconverged nodes need to be tuned in order to address this to maintain stability and maximize the number of possible instances. To do this, you need to set resource constraints for the compute service on hyperconverged nodes.

Resource constraints are configured during an overcloud deployment or upgrade using plan environment files. Although installation procedures for RHOSP are beyond the scope of this course, it is useful to understand how the resource constraints are calculated and set. Default plan samples are available, which include several CPU and memory allocation workload profiles from which to select a match for your requirements.

A workload profile includes the `average_guest_memory_size_in_mb` and `average_guest_cpu_utilization_percentage` parameters, which are used to calculate the `reserved_host_memory` and `cpu_allocation_ratio` settings of the compute service. These values are calculated based on Red Hat recommendations. One or both of those final compute service settings can be overridden manually in the plan environment file.

The `reserved_host_memory` parameter sets how much RAM should be reserved for the Ceph OSD services and per guest instance overhead on hyperconverged nodes. For example, on a node with 256 GB of memory and 10 OSDs that typically consume 3 GB each of memory, allocating 30 GB of memory for Ceph, leaves 226 GB for compute, which could host 113 instances that use 2 GB of memory each.

The `cpu_allocation_ratio` parameter sets the ratio that the compute scheduler should use when choosing which compute node to deploy an instance. By default, this is 16.0 (that is, 16:1). This means that if there are 24 cores on a node, the compute scheduler will schedule enough instances to consume 384 vCPUs on a node before considering the node unable to host any more.

The Hyperconverged Infrastructure Guide includes the appendix *Compute CPU and Memory Calculator* which explains how the Red Hat recommended resource constraint parameters are calculated.

Ceph Backfill

Ceph uses a backfilling and recovery process to rebalance the storage cluster whenever an OSD is removed. This is done to keep multiple copies of the data, according to the placement group policy. These two operations use system resources, so when a Ceph storage cluster is under load, Ceph's performance will drop as Ceph diverts resources to the backfill and recovery process. To maintain acceptable performance of Ceph storage when an OSD is removed, reduce the priority of backfill and recovery operations. The trade-off for reducing the priority is that there are fewer data replicas for a longer period of time, which puts the data at a slightly greater risk.

Similar to the resource constraints, these backfill and recovery operation tuning parameters are set in overcloud deployment plan environment files, and are included here to introduce how hyperconverged nodes are tuned. The three variables to modify are described below:

- The `osd_recovery_max_active` parameter sets the number of active recovery requests per OSD at one time. More requests will accelerate recovery, but the requests place an increased load on the cluster.
- The `osd_max_backfills` parameter sets the maximum number of backfills allowed to or from a single OSD.
- The `osd_recovery_op_priority` parameter sets the priority for recovery operations. It is relative to the `osd_client_op_priority` parameter.

More information on the OSD configurable parameters are found in the *Red Hat Ceph Storage Configuration Guide*.

HOST AGGREGATES

Host aggregates are used to create further partitioning distinctions for nodes within an availability zone. Creating and configuring aggregates requires administrator privileges. Host aggregates are used to assign metadata key-value pairs to groups of nodes. Nodes can be members of multiple host aggregates. The same key-value metadata can be assigned to many aggregates, and aggregates can be configured with as many key-value pairs as needed to define the host selection scenario. When an instance is launched with custom flavor metadata, the metadata is passed to the compute scheduler to perform advanced scheduling selection from available host aggregates in the availability zone. For example, this can be used to define logical groups to control instance deployment or migration by specifying required additional hardware or performance characteristics.

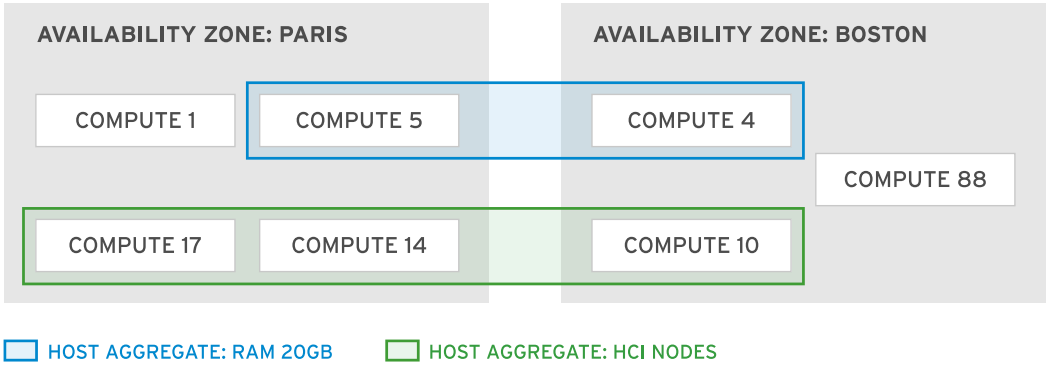


Figure 7.2: Host aggregates

OpenStack users cannot see configured host aggregates, but instead select appropriate flavors to specify the host characteristics to be filtered by the compute scheduler during an instance launch. Administrator privileges are required to configure host aggregates and flavors with matching metadata properties. Set the key-value property on any host aggregate to include the host characteristic property, and then set the same property again on any flavor to be used to request launching to hosts with the required host characteristics.

Availability zones, which are commonly used to specify geographical host groupings, are also a type of host aggregate. Availability zones have a special flag set which allows zones to be seen and selected by normal users to control an instance launch. Administrators can set that flag on any host aggregate to make it visible as an availability zone. One other important difference is that hosts can be members of many aggregates but of only one availability zone.

REFERENCES

OpenStack Administration Guide

https://docs.openstack.org/nova/queens/user/aggregates.html

REFERENCES

For more information, refer to the *Configuring Resource Isolation on Hyper-converged Nodes* chapter in the *Hyper-converged Infrastructure Guide* at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/

For more information, refer to the *Red Hat Ceph Storage Configuration Guide* at https://access.redhat.com/documentation/en-us/red_hat_ceph_storage/3/

LAUNCHING TO A HYPERCONVERGED COMPUTE NODE

In this exercise, you will create an aggregate and a flavor forcing the creation of certain instances on the HCI node.

OUTCOMES

You should be able to create a flavor using aggregate and metadata information to control instance placement, and observe messages in the `nova_scheduler` service log file.

BEFORE YOU BEGIN

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

Run the **`lab computeresources-hci setup`** command. This script ensures that OpenStack services are running and the environment is properly configured for the guided exercise.

```
[student@workstation ~]$ lab computeresources-hci setup
```

1. As the `architect1` user, create a new aggregate called `hci-aggregate`. Add the `computehci0` node to the `hci-aggregate` aggregate.

   1.1. On `workstation`, source the **`/home/student/architect1-finance-rc`** identity environment file to provide administrator access to OpenStack services.

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

   1.2. Use the **`openstack aggregate create`** command to create a new aggregate called `hci-aggregate`.

```
[student@workstation ~(architect1-finance)]$ openstack aggregate create \
hci-aggregate
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2018-10-16T08:21:28.625185 |
| deleted           | False                      |
| deleted_at        | None                       |
| id                | 1                          |
| name              | hci-aggregate              |
| updated_at        | None                       |
+-------------------+----------------------------+
```

   1.3. Use the **`openstack aggregate add host`** command to add the `computehci0` node to the `hci-aggregate` aggregate.

```
[student@workstation ~(architect1-finance)]$ openstack aggregate add host \
hci-aggregate computehci0.overcloud.example.com
+-------------------+---------------------------------------+
| Field             | Value                                 |
+-------------------+---------------------------------------+
| availability_zone | None                                  |
| created_at        | 2018-10-16T08:21:28.000000            |
| deleted           | False                                 |
| deleted_at        | None                                  |
| hosts             | [u'computehci0.overcloud.example.com'] |
| id                | 1                                     |
...
| name              | hci-aggregate                         |
| updated_at        | None                                  |
+-------------------+---------------------------------------+
```

2. Set the metadata for the `hci-aggregate` aggregate.

2.1. Use the **`openstack aggregate set`** command to set the metadata for the `hci-aggregate` aggregate.

```
[student@workstation ~(architect1-finance)]$ openstack aggregate set \
--property computehci=true hci-aggregate
```

2.2. Use the **`openstack aggregate show`** command to show the details of the `hci-aggregate` aggregate.

```
[student@workstation ~(architect1-finance)]$ openstack aggregate show \
hci-aggregate
+-------------------+---------------------------------------+
| Field             | Value                                 |
+-------------------+---------------------------------------+
| availability_zone | None                                  |
| created_at        | 2018-10-16T08:21:28.000000            |
| deleted           | False                                 |
| deleted_at        | None                                  |
| hosts             | [u'computehci0.overcloud.example.com'] |
| id                | 1                                     |
| name              | hci-aggregate                         |
| properties        | computehci='true'                     |
| updated_at        | None                                  |
+-------------------+---------------------------------------+
```

3. Create a new flavor and configure the `computehci` property.

3.1. Use the **`openstack flavor create`** command to create a new flavor called `default-hci`. Configure the flavor to use 1024 MB of RAM, to have a 10 GB disk and to have two vCPUs. Configure the flavor to be public.

```
[student@workstation ~(architect1-finance)]$ openstack flavor create \
--ram 1024 --disk 10 --vcpus 2 --public default-hci
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
```

```
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| disk                       | 10                                   |
| id                         | d159f88a-ccab-4dca-8a91-de16e416792b |
| name                       | default-hci                          |
| os-flavor-access:is_public | True                                 |
| properties                 |                                      |
| ram                        | 1024                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 2                                    |
+----------------------------+--------------------------------------+
```

3.2. Use the **`openstack flavor set`** command to configure the `default-hci` flavor to use the `computehci` property.

```
[student@workstation ~(architect1-finance)]$ openstack flavor set \
default-hci --property computehci=true
```

```
[student@workstation ~(architect1-finance)]$ openstack flavor show default-hci
+----------------------------+--------------------------------------+
| Field                      | Value                                |
+----------------------------+--------------------------------------+
| OS-FLV-DISABLED:disabled   | False                                |
| OS-FLV-EXT-DATA:ephemeral  | 0                                    |
| access_project_ids         | None                                 |
| disk                       | 10                                   |
| id                         | d159f88a-ccab-4dca-8a91-de16e416792b |
| name                       | default-hci                          |
| os-flavor-access:is_public | True                                 |
| properties                 | computehci='true'                    |
| ram                        | 1024                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 2                                    |
+----------------------------+--------------------------------------+
```

4. Use the **`openstack image list`** command to ensure the `hci-image` image exists. Use the **`openstack image show`** command to confirm the size of the image.

```
[student@workstation ~(architect1-finance)]$ openstack image list
+--------------------------------------+-----------------+--------+
| ID                                   | Name            | Status |
+--------------------------------------+-----------------+--------+
| e0ba1c8f-0aa2-4599-a487-a506974b4cb7 | hci-image       | active |
...output omitted...
```

```
[student@workstation ~(architect1-finance)]$ openstack image show \
hci-image --max-width=80
+------------------+-------------------------------------------------------------+
| Field            | Value                                                       |
+------------------+-------------------------------------------------------------+
| checksum         | cbe377ed77279c4a136e05e713db3a50                            |
```

```
| container_format | bare                                                          |
| created_at       | 2018-10-16T08:06:51Z                                          |
| disk_format      | qcow2                                                         |
| file             | /v2/images/e0ba1c8f-0aa2-4599-a487-a506974b4cb7/file          |
| id               | e0ba1c8f-0aa2-4599-a487-a506974b4cb7                          |
| min_disk         | 10                                                            |
| min_ram          | 1024                                                          |
| name             | hci-image                                                     |
| owner            | 3bc7526a7d44471d8cad16a73e2d5871                              |
| properties       | direct_url='rbd://945a02d4-c931-11e8-b4cd-                    |
|                  | 52540001fac8/images/e0ba1c8f-                                 |
|                  | 0aa2-4599-a487-a506974b4cb7/snap', locations='[{u'url':       |
|                  | u'rbd://945a02d4-c931-11e8-b4cd-52540001fac8/images           |
|                  | /e0ba1c8f-0aa2-4599-a487-a506974b4cb7/snap', u'metadata':     |
| protected        | True                                                          |
| schema           | /v2/schemas/image                                             |
| size             | 1621950464                                                    |
| status           | active                                                        |
| tags             |                                                               |
| updated_at       | 2018-10-16T08:07:10Z                                          |
| virtual_size     | None                                                          |
| visibility       | public                                                        |
+------------------+---------------------------------------------------------------+
```

5. Open a new terminal on `workstation`. From `workstation`, log on to the `controller0` node. On `controller0`, configure nova for debugging. Restart the containers and inspect the `nova_scheduler` log files.

   5.1. From a new terminal window on `workstation`, log on to `controller0` as the `heat-admin` user.

```
[student@workstation ~]$ ssh heat-admin@controller0
Warning: Permanently added 'controller0,172.25.250.1' (ECDSA) to the list of known
 hosts.
[heat-admin@controller0 ~]$
```

   5.2. Use the **`sudo -i`** command to gain root privileges. Edit the **`nova.conf`** file and set the `debug` parameter to **`true`**. Restart the `nova_scheduler` and `nova_api` containers. Wait for 45 seconds. Use the **`docker ps`** to ensure the containers are running and healthy.

```
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf \
DEFAULT debug True
```

```
[root@controller0 ~]# docker restart nova_api
nova_api
[root@controller0 ~]# docker restart nova_scheduler
nova_scheduler
```

```
[heat-admin@controller0 ~]$ docker ps \
--format="{{.Names}}\t{{.Status}}" | grep nova
...output omitted...
```

```
nova_api        Up About 31 seconds (healthy)
nova_scheduler  Up About 46 seconds (healthy)
...output omitted...
```

6. Create an instance using the `default-hci` flavor. Ensure the instance is created on the `computehci0` node. Inspect the **`nova-scheduler.log`** log file.

6.1. In the original `workstation` terminal, use the **`openstack server create`** command to create a new instance called `hci-server1`. Use the `default-hci` flavor, the `hci-image` image, and the `finance-network1` network.

```
[student@workstation ~(architect1-finance)]$ openstack server create \
--flavor default-hci \
--image hci-image \
--nic net-id=finance-network1 \
--wait hci-server1
```

6.2. During the creation of `hci-server1`, return to the `controller0` node. Run the **`tail -f`** command on the **`nova-scheduler.log`** file and redirect the output to a file called **`hci.txt`**.

```
[root@controller0 nova]# tail -f \
/var/log/containers/nova/nova-scheduler.log > hci.txt
```

6.3. On `workstation`, confirm that `hci-server1` was created on `computehci0`. Note the ID of `hci-server1` and compare it to the debug entries in the **`hci.txt`** file.

```
[student@workstation ~(architect1-finance)]$ openstack server show \
hci-server1 --max-width=80
+-------------------------------------+----------------------------------------+
| Field                               | Value                                  |
+-------------------------------------+----------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                 |
| OS-EXT-AZ:availability_zone         | nova                                   |
| OS-EXT-SRV-ATTR:host                | computehci0.overcloud.example.com      |
| OS-EXT-SRV-ATTR:hypervisor_hostname | computehci0.overcloud.example.com      |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000002                      |
| OS-EXT-STS:power_state              | Running                                |
| OS-EXT-STS:task_state               | None                                   |
| OS-EXT-STS:vm_state                 | active                                 |
| OS-SRV-USG:launched_at              | 2018-10-16T09:06:00.000000             |
| OS-SRV-USG:terminated_at            | None                                   |
| accessIPv4                          |                                        |
| accessIPv6                          |                                        |
| addresses                           | finance-network1=192.168.1.5           |
| config_drive                        |                                        |
| created                             | 2018-10-16T09:05:31Z                   |
| flavor                              | default-hci (d159f88a-ccab-4dca-       |
|                                     | 8a91-de16e416792b)                     |
| hostId                              | 9bf095a2180a28d194225768846ba007161890 |
|                                     | 1cc37b032d9d44a1be                     |
| id                                  | 9cc87d2c-efd3-45ae-940c-a3f08c590ace   |
| image                               | hci-image (e0ba1c8f-                   |
|                                     | 0aa2-4599-a487-a506974b4cb7)           |
| key_name                            | None                                   |
```

```
| name                                 | hci-server1                                |
| progress                             | 0                                          |
| project_id                           | 7adc3822ee54472b85b2d34cc83f5803           |
| properties                           |                                            |
| security_groups                      | name='default'                             |
| status                               | ACTIVE                                     |
| updated                              | 2018-10-16T09:06:00Z                       |
| user_id                              | 62dc592686267444e8065dd49673d5cfe7868b |
|                                      | 4735edff02bea576d220c938fb                 |
| volumes_attached                     |                                            |
+--------------------------------------+--------------------------------------------+
```

6.4. On the `controller0` node use the **`view`** command to open the **`hci.txt`** Search for the ID of `hci-server1`. Notice the *request ID* for these entries. Entries with the same *request ID* are part of the same request.

```
[root@controller0 nova]# view hci.txt
...output omitted...
2018-10-16 12:20:27.240 1 DEBUG nova.scheduler.filter_scheduler
 [req-5406d5ea-432a-46dc-9b19-042fa769e8b0
 62dc592686267444e8065dd49673d5cfe7868b4735edff02bea576d220c938fb
 7adc3822ee54472b85b2d34cc83f5803 - 04c2c4fea5bb4a839101a89ef9134e2c
 default] Weighed [(computehci0.overcloud.example.com,
 computehci0.overcloud.example.com) ram: 2047MB disk: 106496MB
 io_ops: 0 instances: 0, (compute1.overcloud.example.com,
 compute1.overcloud.example.com) ram: 2047MB disk: 108544MB io_ops: 0 instances:
 0, (compute0.overcloud.example.com, compute0.overcloud.example.com) ram: 2047MB
 disk: 108544MB io_ops: 0 instances: 0] _get_sorted_hosts /usr/lib/python2.7/site-
packages/nova/scheduler/filter_scheduler.py:424
2018-10-16 12:20:27.241 1 DEBUG nova.scheduler.utils
 [req-5406d5ea-432a-46dc-9b19-042fa769e8b0
 62dc592686267444e8065dd49673d5cfe7868b4735edff02bea576d220c938fb
 7adc3822ee54472b85b2d34cc83f5803 - 04c2c4fea5bb4a839101a89ef9134e2c
 default] Attempting to claim resources in the placement API for
 instance 9cc87d2c-efd3-45ae-940c-a3f08c590ace claim_resources /usr/lib/python2.7/
site-packages/nova/scheduler/utils.py:786
2018-10-16 12:20:27.283 1 DEBUG nova.scheduler.filter_scheduler
 [req-5406d5ea-432a-46dc-9b19-042fa769e8b0
 62dc592686267444e8065dd49673d5cfe7868b4735edff02bea576d220c938fb
 7adc3822ee54472b85b2d34cc83f5803 - 04c2c4fea5bb4a839101a89ef9134e2c
 default] Selected host: (computehci0.overcloud.example.com,
 computehci0.overcloud.example.com) ram: 2047MB disk: 106496MB io_ops: 0
 instances: 0 _consume_selected_host /usr/lib/python2.7/site-packages/nova/
scheduler/filter_scheduler.py:325
2018-10-16 12:20:27.283 1 DEBUG oslo_concurrency.lockutils
 [req-5406d5ea-432a-46dc-9b19-042fa769e8b0
 62dc592686267444e8065dd49673d5cfe7868b4735edff02bea576d220c938fb
 7adc3822ee54472b85b2d34cc83f5803 - 04c2c4fea5bb4a839101a89ef9134e2c default] Lock
 "(u'computehci0.overcloud.example.com', u'computehci0.overcloud.example.com')"
 acquired by "nova.scheduler.host_manager._locked" :: waited 0.000s inner /usr/
lib/python2.7/site-packages/oslo_concurrency/lockutils.py:273
...output omitted...
```

Cleanup

On `workstation`, run the **`lab computeresources-hci cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab computeresources-hci cleanup
```

This concludes the guided exercise.

ADMINISTERING COMPUTE NODES

OBJECTIVES

After completing this section, students should be able to:

- Discuss the migration process.
- Migrate an instance.
- Enable and disable a compute node.

INTRODUCTION TO MIGRATION AND EVACUATION

Migration is the process of moving a server instance from one compute node to another. There are three different kinds of migration in OpenStack.

In a *Cold Migration* the administrator shuts down the instance if it is in a running state. A copy of the image is sent from the source compute node to the destination compute node. The instance is reassigned to the destination host in the database and restarted.

Live Migration moves the instance from one host to another without being powered off. The instance is unaware that it is being moved. Active client connections are maintained and work can be performed during the migration. Applications and users communicating with the applications and services on this instance should not detect interruption. Disks must also be transferred and attached to the destination virtual machine. There are two methods for disk transfer: *block based* and *shared storage*.

Evacuate normally occurs when the compute node goes into failure or goes into shutdown mode. It is possible to evacuate a running compute node but it is not considered a normal use case for the evacuation process. Instances are rebuilt on another compute node. Instances keep their name, their UUID, network addresses, and any other allocated resources. The evacuation process is destructive if the instance disks are not on shared storage or block storage volumes. OpenStack chooses which compute nodes will receive the evacuated instances.

Instance Migration

An instance can be migrated while running using the **`openstack server migrate`** command with the **`--live`** option. To migrate an instance that is stopped, use the same command without the **`--live`** option. In both cases, the administrator, using the **`--target`**, informs OpenStack of the destination compute node.

The **`--shared-migration`** and **`--block-migration`** options are used to manage disk transfer. By default, the **`--shared-migration`** option is implemented by OpenStack. Red Hat recommended practice for overcloud deployment is to install shared storage using a Ceph RBD storage cluster. Instances can be deployed using the Block Storage service volume-based transient or persistent disks instead of using the compute service ephemeral storage, but compute nodes configured with different back ends require block-based migration.

To list the servers on a specific compute node, use the **`openstack server list`** command.

```
(overcloud) [stack@director ~]$ openstack server list \
--host compute0.overcloud.example.com --all-projects \
-c ID -c Name
+--------------------------------------+-----------------+
```

```
| ID                                   | Name            |
+--------------------------------------+-----------------+
| e1060cd2-154b-48fc-930e-36ac92ad00ae | finance-server3 |
| d70a2397-77f3-4fa4-8af4-1404ac4f2b0d | finance-server2 |
| a181bb99-e8b1-4d74-8e46-37d46a7d5024 | finance-server1 |
+--------------------------------------+-----------------+
```

A server can be live migrated from one compute host to another using the **`openstack server migrate`** command. Notice the absence of either the **`--shared-migration`** or the **`--block-migration`** options. OpenStack, by default, opts for shared storage migration.

```
(overcloud) [stack@director ~]$ openstack server migrate \
a181bb99-e8b1-4d74-8e46-37d46a7d5024 \
--live compute1.overcloud.example.com
```

Instance Evacuation

To perform an evacuation of a compute node using the **`nova host-evacuate`** command it must first be disabled. Use the **`openstack compute service set`** command with the **`--disable`** option to disable the compute node.

```
(overcloud) [stack@director ~]$ openstack compute service set \
    compute0.overcloud.example.com nova-compute --disable
```

```
(overcloud) [stack@director ~]$ openstack compute service list \
-c Binary -c Host -c Status
...output omitted...
| nova-compute     | computehci0.overcloud.example.com | enabled  |
| nova-compute     | compute0.overcloud.example.com    | disabled |
| nova-compute     | compute1.overcloud.example.com    | enabled  |
+------------------+-----------------------------------+----------+
```

The disabled compute node can then be evacuated using the **`nova host-evacuate`** command.

```
(overcloud) [stack@director ~]$ nova host-evacuate compute0.overcloud.example.com
```

To force the evacuation of servers from a running compute node to another compute node, use the **`nova host-evacuate-live`** command. If the **`nova host-evacuate`** command is used on a running compute node, it returns the following error:

```
+-------------+------------------------+----------------------------------+
| Server UUID | Live Migration Accepted | Error Message                    |
+-------------+------------------------+----------------------------------+
| 8186...f6e7 | False                  |Error while evacuating instance:  |
|             |                        |Compute service of                |
|             |                        |compute1.overcloud.example.com    |
|             |                        |is still in use. (HTTP 400)       |
|             |                        |(Request-ID: req-3ead...bbc7)     |
+-------------+------------------------+----------------------------------+
```

When an instance has been accepted for migration, either by using the **`nova host-evacuate`** command on a disabled compute node or by using the **`nova host-evacuate-live`** command on a running compute node, the following message is displayed:

```
+--------------------------------------+------------------------+---------------+
| Server UUID                          | Live Migration Accepted | Error Message |
+--------------------------------------+------------------------+---------------+
| 81863f14-fec4-4649-9fcc-54017e48f6e7 | True                   |               |
+--------------------------------------+------------------------+---------------+
```

The instances being moved have a status of **MIGRATING** while they are being moved. After they have been moved to the new compute node their status changes to **ACTIVE**.

```
+-----------------+-----------+
| Name            | Status    |
+-----------------+-----------+
| finance-server3 | MIGRATING |
+-----------------+-----------+
```

```
+-----------------+--------+
| Name            | Status |
+-----------------+--------+
| finance-server3 | ACTIVE |
+-----------------+--------+
```

STORAGE MIGRATION

Shared Storage Migration

With the shared storage method, if both the source and destination compute nodes connect to and have sufficient access privileges for the same shared storage locations containing the migrating instance's disks, then no physical disk movement occurs. The source compute node stops using the disks while the destination compute node takes over disk activity.

When the **`openstack server migrate`** command is issued, the source sends an instance's memory content to the destination. During the transfer process, the memory pages on the source host are still being modified in real time. The source host tracks the memory pages that were modified during the transfer and retransmits them after the initial bulk transfer is completed. The instance's memory content must be transferred faster than memory pages are written on the source virtual machine. After all re-transmission is complete, an identical instance is started on the destination host. In parallel, the virtual network infrastructure redirects the network traffic.

Live migration using block storage uses a similar process as shared storage live migration. However, with block storage, disk content is copied before the memory content is transferred, making live migration with shared storage quicker and more efficient.

Live Migration Configuration Options

The following is a list of the live migration configuration options available for libvirt in **`/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf`** and their default values. This configuration file is found on all compute and HCI nodes.

Live Migration Configuration Options

PARAMETERS	DESCRIPTION
`live_migration_retry_count = 30`	Number of retries needed in live migration.

PARAMETERS	DESCRIPTION
`max_concurrent_live_migrations = 1`	Maximum number of concurrent live migrations to run.
`live_migration_bandwidth = 0`	Maximum bandwidth in MiB/s. If set to 0, a suitable value is chosen automatically.
`live_migration_completion_timeout = 800`	Timeout value in seconds for successful migration to complete before aborting the operation. The value depends on the size of the instance.
`live_migration_downtime = 500`	Maximum permitted downtime, in milliseconds.
`live_migration_downtime_delay = 75`	Time to wait, in seconds, between each step increase of the migration downtime.
`live_migration_downtime_steps = 10`	Number of incremental steps to reach max downtime value.
`live_migration_flag = VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER, VIR_MIGRATE_LIVE, VIR_MIGRATE_TUNNELLED`	Migration flags to be set for live migration.
`live_migration_progress_timeout = 150`	Time to wait, in seconds, for migration to make progress in transferring data before aborting the operation.
`live_migration_uri = qemu+tcp://%s/system`	Migration target URI.

When an instance is nearing the end of the memory copy it is paused for a period of time. This downtime allows the remaining pages to be copied without interference from the memory writes. The value for this period of time depends on the instance size. By default it is 500 milliseconds. If the memory copy is not making the expecting progress the time increases gradually. The downtime algorithm is influenced using the `live_migration_downtime`, `live_migration_downtime_delay`, and `live_migration_downtime_steps` parameters.

Block Migration

Block-based live migration is the alternative method used when shared storage is not implemented. When the source and destination compute nodes do not share common access storage, the root, ephemeral, swap, and persistent volumes must be transferred to the storage location used by the destination compute node. When performance is a primary focus, block-based live migration should be avoided. Instead, implement shared storage structures across common networks where live migration occurs regularly.

Block-based Live Migration Use Cases

Red Hat recommended practice for overcloud deployment is to install shared storage using a Ceph RBD storage cluster, but earlier use cases offered various configurations for managing instance disks:

- Although **`/var/lib/nova/instances`** can be shared across compute nodes using `GlusterFS` or `NFS`, the default configuration had each compute node maintaining disk storage for each instance scheduled to their hypervisor. An instance rescheduled or redeployed to another hypervisor would cause a duplicate set of that instance's disks to be deployed on the new compute node.

- Different compute nodes, even when operating in the same networks, can be connected to different storage arrays, Red Hat Virtualization data stores, or other back-end storage subsystems.

- Instances can be deployed using the Block Storage service volume-based transient or persistent disks instead of using the compute service ephemeral storage, but compute nodes configured with different back ends require block-based migration.

Implementation Requirements for Block-based Live Migration

There are specific requirements for implementing block-based live migration:

- Both source and destination compute nodes must be located in the same subnet.

- Both compute nodes must use the same processor type.

- All controller and compute nodes must have consistent name resolution for all other nodes.

- The UID and GID of the `nova` and `libvirt` users must be identical on all compute nodes.

- Compute nodes must be using KVM with libvirt, which is expected when using Red Hat OpenStack Platform. The KVM with libvirt platform has the best coverage of features and stability for live migration.

- The permissions and system access of local directories must be consistent across all nodes.

- libvirt must be able to securely communicate between nodes.

- Consistent multipath device naming must be used on both the source and destination compute nodes. Instances expect to resolve multipath device names similarly in both locations.

Troubleshooting

When migration fails or takes too long, check the activity in the **`nova-compute.log`** log files on both the source and the destination `compute` nodes. The *scheduler*, *api* and *conductor* log files are now found on the `controller` nodes.

- **`/var/log/containers/nova/nova-compute.log`**
- **`/var/log/containers/nova/nova-api.log`**
- **`/var/log/containers/nova/nova-scheduler.log`**
- **`/var/log/containers/nova/nova-conductor.log`**

REFERENCES

OpenStack Documentation: Live Migrate Instances

https://docs.openstack.org/nova/queens/admin/live-migration-usage.html

ADMINISTERING COMPUTE NODES

In this exercise, you will migrate an instance, disable and enable compute nodes, and evacuate a compute node.

OUTCOMES

You should be able to enable and disable compute nodes, migrate instances, and evacuate an entire compute node.

BEFORE YOU BEGIN

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

Run the **`lab computeresources-migrate setup`** command. This script ensures that OpenStack services are running and the environment is properly configured for the guided exercise.

```
[student@workstation ~]$ lab computeresources-migrate setup
```

1. On `workstation`, source the **`/home/student/architect1-finance-rc`** environment identity file and then list the compute services.

   1.1. Source the **`/home/student/architect1-finance-rc`** file.

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

   1.2. Use the **`openstack compute service list`** command to list the Host, Zone, and Status columns for each of the service nodes. Ensure that the status of the `compute` nodes is **`enabled`**.

```
[student@workstation ~(architect1-finance)]$ openstack compute service list \
-c Host -c Zone -c Status
+-----------------------------------+----------+---------+
| Host                              | Zone     | Status  |
+-----------------------------------+----------+---------+
| controller0.overcloud.example.com | internal | enabled |
| controller0.overcloud.example.com | internal | enabled |
| controller0.overcloud.example.com | internal | enabled |
| computehci0.overcloud.example.com | nova     | enabled |
| compute0.overcloud.example.com    | nova     | enabled |
| compute1.overcloud.example.com    | nova     | enabled |
+-----------------------------------+----------+---------+
```

2. Use the **`openstack server list`** command to list the servers on both `compute0` and `compute1`.

```
[student@workstation ~(architect1-finance)]$ openstack server list \
--host compute0.overcloud.example.com --all-projects \
-c ID -c Name
+--------------------------------------+-----------------+
| ID                                   | Name            |
+--------------------------------------+-----------------+
| e1060cd2-154b-48fc-930e-36ac92ad00ae | finance-server3 |
| d70a2397-77f3-4fa4-8af4-1404ac4f2b0d | finance-server2 |
| a181bb99-e8b1-4d74-8e46-37d46a7d5024 | finance-server1 |
+--------------------------------------+-----------------+
[student@workstation ~(architect1-finance)]$ openstack server list \
--host compute1.overcloud.example.com --all-projects \
-c ID -c Name

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

Notice that all three instances are hosted on `compute0`.

3. Migrate `finance-server1` from `compute0` to `compute1`. Verify that the server has been migrated.

3.1. Use the **`openstack server migrate`** command to migrate `finance-server1`. The ID for `finance-server1` can be found in the output of the previous step.

```
[student@workstation ~(architect1-finance)]$ openstack server migrate \
a181bb99-e8b1-4d74-8e46-37d46a7d5024 \
--live compute1.overcloud.example.com
```

3.2. Use the **`openstack server list`** command to verify that `finance-server1` is now hosted on `compute1`. The migration may take a minute so retry this command if required.

```
[student@workstation ~(architect1-finance)]$ openstack server list \
--host compute1.overcloud.example.com --all-projects \
-c ID -c Name
+--------------------------------------+-----------------+
| ID                                   | Name            |
+--------------------------------------+-----------------+
| a181bb99-e8b1-4d74-8e46-37d46a7d5024 | finance-server1 |
+--------------------------------------+-----------------+
```

4. Disable the `nova-compute` service on `compute0`. Evacuate the remaining servers to `compute1`. Verify the servers are migrated to `compute1`. Lastly, enable the `nova-compute` service on `compute0`.

4.1. Use the **`openstack compute service set`** command to disable the `nova-compute` service on `compute0`.

```
[student@workstation ~(architect1-finance)]$ openstack compute service set \
```

```
compute0.overcloud.example.com nova-compute --disable
```

4.2. Use the **`openstack compute service list`** command to verify that `compute0` is disabled.

```
[student@workstation ~(architect1-finance)]$ openstack compute service list \
-c Binary -c Host -c Status
...output omitted...
| nova-compute     | computehci0.overcloud.example.com | enabled  |
| nova-compute     | compute0.overcloud.example.com    | disabled |
| nova-compute     | compute1.overcloud.example.com    | enabled  |
+-----------------+-----------------------------------+----------+
```

4.3. Use the **`nova host-evacuate-live`** command to evacuate the remaining servers from `compute0`.

```
[student@workstation ~(architect1-finance)]$ nova host-evacuate-live \
compute0
+--------------------------------------+-------------------------+---------------+
| Server UUID                          | Live Migration Accepted | Error Message |
+--------------------------------------+-------------------------+---------------+
| d70a2397-77f3-4fa4-8af4-1404ac4f2b0d | True                    |               |
| e1060cd2-154b-48fc-930e-36ac92ad00ae | True                    |               |
+--------------------------------------+-------------------------+---------------+
```

4.4. Use the **`openstack server list`** command to verify that the instances have been migrated to `compute1`. It may take a minute to complete the migration. Retry the command until all servers are migrated.

```
[student@workstation ~(architect1-finance)]$ openstack server list \
--host compute1.overcloud.example.com --all-projects \
-c ID -c Name
+--------------------------------------+-----------------+
| ID                                   | Name            |
+--------------------------------------+-----------------+
| e1060cd2-154b-48fc-930e-36ac92ad00ae | finance-server3 |
| d70a2397-77f3-4fa4-8af4-1404ac4f2b0d | finance-server2 |
| a181bb99-e8b1-4d74-8e46-37d46a7d5024 | finance-server1 |
+--------------------------------------+-----------------+
```

4.5. Use the **`openstack compute service set`** command to enable the `nova-compute` service on `compute0`. Use the **`openstack compute service list`** command to verify the status of the service.

```
[student@workstation ~(architect1-finance)]$ openstack compute service set \
compute0.overcloud.example.com nova-compute --enable
[student@workstation ~(architect1-finance)]$ openstack compute service list \
-c Host -c Status
...output omitted...
| computehci0.overcloud.example.com | enabled |
| compute0.overcloud.example.com    | enabled |
| compute1.overcloud.example.com    | enabled |
+-----------------------------------+---------+
```

Cleanup

On `workstation`, run the **`lab computeresources-migrate cleanup`** script to clean up the resources created in this exercise.

```
[student@workstation ~]$ lab computeresources-migrate cleanup
```

This concludes the guided exercise.

MANAGING COMPUTE RESOURCES

PERFORMANCE CHECKLIST

In this lab, you will manage compute resources.

OUTCOMES

You should be able to:

- Deploy an instance to the HCI node.
- Perform an evacuation.
- Perform a block migration.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab computeresources-review setup`** command. This script creates the resources required to perform the exercise.

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

1. One of the compute nodes is currently powered off. Find the powered off compute node and evacuate the instances to another node. Power on the compute node.
2. As the `architect1` user, create a new aggregate called `hci-aggregate`. Add the `computehci0` node to the `hci-aggregate` aggregate. Set the metadata for the `hci-aggregate` aggregate. Create a new flavor called `default-hci` and configure the `computehci` property. Launch a new instance called `hci-server1`. Use the `default-hci` flavor, the `hci-image` image, and the `production-network1` network. Confirm that the instance is launched on the `computehci0` compute node.
3. Configure nova to use block storage. Restart the `nova_compute` container on both `compute0` and `compute1`. Confirm the health of the container. As the `architect1` user, create an instance on `compute0`. Use the `default` flavor, the `rhel7` image, and the `production-network1` network. Migrate the instance to `compute1` using block migration.

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab computeresources-review cleanup
```

This concludes the lab.

MANAGING COMPUTE RESOURCES

PERFORMANCE CHECKLIST

In this lab, you will manage compute resources.

OUTCOMES

You should be able to:

- Deploy an instance to the HCI node.
- Perform an evacuation.
- Perform a block migration.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab computeresources-review setup`** command. This script creates the resources required to perform the exercise.

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

1. One of the compute nodes is currently powered off. Find the powered off compute node and evacuate the instances to another node. Power on the compute node.

   1.1. From `workstation` log in to `director` as the `stack` user. Use the **`openstack baremetal node list`** command to list the bare-metal nodes to discover which node is powered off.

```
[student@workstation ~]$ ssh stack@director
```

```
(undercloud) [stack@director ~]$ openstack baremetal node list \
-c Name -c "Power State"
+-------------+-------------+
| Name        | Power State |
+-------------+-------------+
| controller0 | power on    |
| compute0    | power off   |
| computehci0 | power on    |
| compute1    | power on    |
| ceph0       | power on    |
+-------------+-------------+
```

   1.2. Source the **`/home/stack/overcloudrc`** environment identity file. Use the **`openstack server list`** command to list the instances on the *powered off* compute node. Evacuate the instances from the *powered off* compute node, using the **`nova host-evacuate`** command.

```
(undercloud) [stack@director ~]$ source ~/overcloudrc
(overcloud) [stack@director ~]$ openstack server list \
--host compute0.overcloud.example.com --all-projects \
-c ID -c Name
+--------------------------------------+--------------------+
| ID                                   | Name               |
+--------------------------------------+--------------------+
| 066bf13c-219c-45c6-9525-676befbdd50f | production-server3 |
| d094a3d6-95fa-4d75-a027-e04f1f37dc60 | production-server2 |
| 382f9f07-dcb2-4b26-82f2-e38a046de8a4 | production-server1 |
+--------------------------------------+--------------------+
```

```
(overcloud) [stack@director ~]$ nova host-evacuate compute0
+--------------------------------------+-------------------+---------------+
| Server UUID                          | Evacuate Accepted | Error Message |
+--------------------------------------+-------------------+---------------+
| 382f9f07-dcb2-4b26-82f2-e38a046de8a4 | True              |               |
| d094a3d6-95fa-4d75-a027-e04f1f37dc60 | True              |               |
| 066bf13c-219c-45c6-9525-676befbdd50f | True              |               |
+--------------------------------------+-------------------+---------------+
```

1.3. Ensure that the instances have been evacuated to another compute node. The migration may take a minute or two.

```
(overcloud) [stack@director ~]$ openstack server list \
--host compute0.overcloud.example.com --all-projects \
-c ID -c Name

(overcloud) [stack@director ~]$ openstack server list \
--host compute1.overcloud.example.com --all-projects \
-c ID -c Name -c Status
+--------------------------------------+--------------------+--------+
| ID                                   | Name               |        |
+--------------------------------------+--------------------+--------+
| 066bf13c-219c-45c6-9525-676befbdd50f | production-server3 | ACTIVE |
| d094a3d6-95fa-4d75-a027-e04f1f37dc60 | production-server2 | ACTIVE |
| 382f9f07-dcb2-4b26-82f2-e38a046de8a4 | production-server1 | ACTIVE |
+--------------------------------------+--------------------+--------+
```

1.4. Source the **/home/stack/stackrc** environment identity file. Use the **openstack baremetal node** command to power up the compute node. Confirm that the `compute0` baremetal node is powered on.

```
(overcloud) [stack@director ~]$ source ~/stackrc
(undercloud) [stack@director ~]$ openstack baremetal node power on compute0
(undercloud) [stack@director ~]$ openstack baremetal node list \
-c Name -c "Power State"
+-------------+-------------+
| Name        | Power State |
+-------------+-------------+
| controller0 | power on    |
| compute0    | power on    |
| computehci0 | power on    |
```

```
| compute1     | power on     |
| ceph0        | power on     |
+-------------+-------------+
```

1.5. The undercloud compute service configuration might prevent it from being aware of node power state changes in virtual environments, such as this classroom. When you use the **`openstack baremetal`** command to power on a node, the undercloud compute service may not agree with the baremetal service about the state of the node. Use the **`openstack server start`** command to align the undercloud compute service with the baremetal service. This ensures that `compute0` will be available to schedule instances on.

```
(undercloud) [stack@director ~]$ openstack server list -c Name -c Status
+-------------+---------+
| Name        | Status  |
+-------------+---------+
| compute1    | ACTIVE  |
| compute0    | SHUTOFF |
| computehci0 | ACTIVE  |
| controller0 | ACTIVE  |
| ceph0       | ACTIVE  |
+-------------+---------+
(undercloud) [stack@director ~]$ openstack server start compute0
(undercloud) [stack@director ~]$ openstack server list -c Name -c Status
+-------------+--------+
| Name        | Status |
+-------------+--------+
| compute1    | ACTIVE |
| compute0    | ACTIVE |
| computehci0 | ACTIVE |
| controller0 | ACTIVE |
| ceph0       | ACTIVE |
+-------------+--------+
```

1.6. Source the **`/home/stack/overcloudrc`** environment file, then use the **`openstack compute service list`** command to ensure that the `nova_compute` service is **`up`** on `compute0`. Exit the `director` node.

```
(undercloud) [stack@director ~]$ source ~/overcloudrc
(overcloud) [stack@director ~]$ openstack compute service \
list -c Binary -c Host -c State
+------------------+----------------------------------+-------+
| Binary           | Host                             | State |
+------------------+----------------------------------+-------+
| nova-conductor   | controller0.overcloud.example.com | up    |
| nova-compute     | compute1.overcloud.example.com    | up    |
| nova-compute     | computehci0.overcloud.example.com | up    |
| nova-compute     | compute0.overcloud.example.com    | up    |
| nova-consoleauth | controller0.overcloud.example.com | up    |
| nova-scheduler   | controller0.overcloud.example.com | up    |
+------------------+----------------------------------+-------+
(overcloud) [stack@director ~]$ exit
logout
Connection to director closed.
[student@workstation ~]$
```

2. As the `architect1` user, create a new aggregate called `hci-aggregate`. Add the `computehci0` node to the `hci-aggregate` aggregate. Set the metadata for the `hci-aggregate` aggregate. Create a new flavor called `default-hci` and configure the `computehci` property. Launch a new instance called `hci-server1`. Use the `default-hci` flavor, the `hci-image` image, and the `production-network1` network. Confirm that the instance is launched on the `computehci0` compute node.

   2.1. On `workstation`, source the **`/home/student/architect1-production-rc`** identity environment file to provide administrator access to OpenStack services.

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

   2.2. Use the **`openstack aggregate create`** command to create a new aggregate called `hci-aggregate`.

```
[student@workstation ~(architect1-production)]$ openstack aggregate create \
hci-aggregate
+-------------------+----------------------------+
| Field             | Value                      |
+-------------------+----------------------------+
| availability_zone | None                       |
| created_at        | 2018-11-19T15:07:51.605242 |
| deleted           | False                      |
| deleted_at        | None                       |
| id                | 1                          |
| name              | hci-aggregate              |
| updated_at        | None                       |
+-------------------+----------------------------+
```

   2.3. Use the **`openstack aggregate add host`** command to add the `computehci0` node to the `hci-aggregate` aggregate.

```
[student@workstation ~(architect1-production)]$ openstack aggregate add host \
hci-aggregate computehci0.overcloud.example.com
+-------------------+---------------------------------------+
| Field             | Value                                 |
+-------------------+---------------------------------------+
| availability_zone | None                                  |
| created_at        | 2018-11-19T15:07:51.000000            |
| deleted           | False                                 |
| deleted_at        | None                                  |
| hosts             | [u'computehci0.overcloud.example.com'] |
| id                | 1                                     |
| metadata          | {}                                    |
| name              | hci-aggregate                         |
| updated_at        | None                                  |
+-------------------+---------------------------------------+
```

   2.4. Use the **`openstack aggregate set`** command to set the metadata for the `hci-aggregate` aggregate.

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

```
--property computehci=true hci-aggregate
```

2.5. Use the **`openstack aggregate show`** command to show the details of the `hci-aggregate` aggregate.

```
[student@workstation ~(architect1-production)]$ openstack aggregate show \
hci-aggregate
+-------------------+---------------------------------------+
| Field             | Value                                 |
+-------------------+---------------------------------------+
| availability_zone | None                                  |
| created_at        | 2018-11-19T15:07:51.000000            |
| deleted           | False                                 |
| deleted_at        | None                                  |
| hosts             | [u'computehci0.overcloud.example.com'] |
| id                | 1                                     |
| name              | hci-aggregate                         |
| properties        | computehci='true'                     |
| updated_at        | None                                  |
+-------------------+---------------------------------------+
```

2.6. Use the **`openstack flavor create`** command to create a new flavor called `default-hci`. Configure the flavor to use 1024 MB of RAM, to have a 10 GB disk, and to have two vCPUs. Configure the flavor to be public.

```
[student@workstation ~(architect1-production)]$ openstack flavor create \
--ram 1024 --disk 10 --vcpus 2 --public default-hci
+----------------------------+-------------------------------------+
| Field                      | Value                               |
+----------------------------+-------------------------------------+
| OS-FLV-DISABLED:disabled   | False                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                   |
| disk                       | 10                                  |
| id                         | 4f919950-d803-4da4-8a9c-857144498bb7 |
| name                       | default-hci                         |
| os-flavor-access:is_public | True                                |
| properties                 |                                     |
| ram                        | 1024                                |
| rxtx_factor                | 1.0                                 |
| swap                       |                                     |
| vcpus                      | 2                                   |
+----------------------------+-------------------------------------+
```

2.7. Use the **`openstack flavor set`** command to configure the `default-hci` flavor to use the `computehci` property.

```
[student@workstation ~(architect1-production)]$ openstack flavor set \
default-hci --property computehci=true
```

```
[student@workstation ~(architect1-production)]$ openstack flavor show default-hci
+----------------------------+-------------------------------------+
| Field                      | Value                               |
+----------------------------+-------------------------------------+
| OS-FLV-DISABLED:disabled   | False                               |
| OS-FLV-EXT-DATA:ephemeral  | 0                                   |
```

```
| access_project_ids         | None                                 |
| disk                       | 10                                   |
| id                         | 4f919950-d803-4da4-8a9c-857144498bb7 |
| name                       | default-hci                          |
| os-flavor-access:is_public | True                                 |
| properties                 | computehci='true'                    |
| ram                        | 1024                                 |
| rxtx_factor                | 1.0                                  |
| swap                       |                                      |
| vcpus                      | 2                                    |
+----------------------------+--------------------------------------+
```

2.8. Create an instance using the `default-hci` flavor. Verify that the instance is created on the `computehci0` node.

```
[student@workstation ~(architect1-production)]$ openstack server create \
--flavor default-hci \
--image hci-image \
--nic net-id=production-network1 \
--wait hci-server1
+-------------------------------------+--------------------------------------+
| Field                               | Value                                |
+-------------------------------------+--------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                               |
| OS-EXT-AZ:availability_zone         | nova                                 |
| OS-EXT-SRV-ATTR:host                | computehci0.overcloud.example.com    |
| OS-EXT-SRV-ATTR:hypervisor_hostname | computehci0.overcloud.example.com    |
| OS-EXT-SRV-ATTR:instance_name       | instance-0000000d                    |
...output omitted...
```

3. Configure nova to use block storage. Restart the `nova_compute` container on both `compute0` and `compute1`. Confirm the health of the container. As the `architect1` user, create an instance on `compute0`. Use the `default` flavor, the `rhel7` image, and the `production-network1` network. Migrate the instance to `compute1` using block migration.

3.1. On both `compute0` and `compute1`, configure nova to use block storage. Use the **`sudo -i`** command to change to the `root` user. In the **`/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf`** file, comment out the `images_type=rbd` line.

```
[student@workstation ~(architect1-production)]$ ssh compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]#
```

```
[root@compute0 ~]# vim \
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf
#images_type=rbd
```

```
[root@compute0 ~]# docker restart nova_compute
nova_compute
[root@compute0 ~]# docker ps --format="{{.Names}}\t{{.Status}}" \
| grep nova_compute
```

```
nova_compute    Up 55 seconds (healthy)
```

```
[root@compute0 ~]# exit
[heat-admin@compute0 ~]$ exit
logout
Connection to compute0 closed.
[student@workstation ~(architect1-production)]$
```

3.2. Repeat the above steps on `compute1`.

```
[student@workstation ~(architect1-production)]$ ssh compute1
Warning: Permanently added 'compute1,172.25.250.12' (ECDSA) to the list of known
 hosts.
[heat-admin@compute1 ~]$ sudo -i
[root@compute1 ~]#
```

```
[root@compute1 ~]# vim \
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf
#images_type=rbd
```

```
[root@compute1 ~]# docker restart nova_compute
nova_compute
[root@compute1 ~]# docker ps --format="{{.Names}}\t{{.Status}}" \
| grep nova_compute
nova_compute    Up 55 seconds (healthy)
```

```
[root@compute1 ~]# exit
[heat-admin@compute1 ~]$ exit
logout
Connection to compute1 closed.
[student@workstation ~(architect1-production)]$
```

3.3. Create an instance called `production-server10` on `compute1`. Use the `default` flavor, the `rhel7` image, and the `production-network1` network.

```
[student@workstation ~(architect1-production)]$ openstack server create \
--flavor default \
--image rhel7 \
--nic net-id=production-network1 \
--availability-zone nova:compute1.overcloud.example.com \
--wait production-server10
+-------------------------------------+--------------------------------+
| Field                               | Value                          |
+-------------------------------------+--------------------------------+
| OS-DCF:diskConfig                   | MANUAL                         |
| OS-EXT-AZ:availability_zone         | nova                           |
| OS-EXT-SRV-ATTR:host                | compute1.overcloud.example.com |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute1.overcloud.example.com |
```

```
...output omitted...
```

3.4. Use the **`openstack server migrate`** command to migrate `production-server10` to `compute0`.

```
[student@workstation ~(architect1-production)]$ openstack server migrate \
--live compute0.overcloud.example.com \
--block-migration production-server10
```

```
[student@workstation ~(architect1-production)]$ openstack server show \
production-server10
+-------------------------------------+----------------------------------------+
| Field                               | Value                                  |
+-------------------------------------+----------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                 |
| OS-EXT-AZ:availability_zone         | nova                                   |
| OS-EXT-SRV-ATTR:host                | compute0.overcloud.example.com         |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute0.overcloud.example.com         |
...output omitted...
```

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab computeresources-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The instance launch process.
- The scheduler process receives resource requests from the conductor. The request is then sent to the placement service which runs a query to find available compute nodes that can satisfy the resource requirements. The list is then sent back to the scheduler service. The scheduler uses filters to refine the search and creates a ranked list of compute nodes. The scheduler then chooses the first one from the list and the conductor attempts to build the instance.
- Red Hat OpenStack Platform included the placement service in version 13. A new RESTful API is implemented to address the problems of shared storage. Generic resource pools solve the problem of incorrect capacity and usage information.
- Hyperconverged nodes combine both compute and storage on the same node. Ceph is always used as the storage component in hyperconverged nodes. Hyperconverged storage is cheaper and more flexible than standard storage nodes.
- Resource constraints are configured to ensure that Ceph and Compute services do not interfere with each other.
- Migration is the process of moving a server instance from one compute node to another. Evacuation normally occurs when a compute node goes into failure or shutdown mode. In this case all instances on the compute node are moved to another. Instances can be migrated while running either using shared or block storage.

CHAPTER 8

AUTOMATING CLOUD APPLICATIONS

GOAL

Implement complex and scalable applications using orchestration.

OBJECTIVES

- Explain the orchestration architecture required to deploy application stacks.
- Write templates using the Heat Orchestration Template (HOT) language to deploy application stacks.

SECTIONS

- Creating Application Stacks (and Quiz)
- Writing Heat Orchestration Templates (and Guided Exercise)

LAB

- Automating Cloud Applications

CREATING APPLICATION STACKS

OBJECTIVES

After completing this section, students should be able to explain the orchestration architecture required to deploy application stacks.

HEAT ORCHESTRATION AND SERVICES

When managing an OpenStack infrastructure, using scripts can make it difficult to create and manage all the infrastructural resources. Even version control and change tracking in the infrastructure can be challenging. Replicating production environments across multiple developments and testing environments becomes much harder.

The orchestration service (heat) provides developers and system administrators an easy and repeatable way to create and manage a collection of related OpenStack services. The orchestration service deploys OpenStack resources in an orderly and predictable fashion. The user creates an orchestration template to describe OpenStack resources and runtime parameters required to execute an application. The orchestration service handles the ordering of the deployment of these OpenStack resources and resolves any dependencies. When provisioning your infrastructure with the orchestration service, the orchestration template describes the resources to be provisioned and their settings. Because the templates are text files, you can use a version control system to control the template versions to track changes to infrastructure.

UNDERSTANDING THE ORCHESTRATION SERVICE ARCHITECTURE

An orchestration stack is a collection of multiple infrastructure resources deployed and managed through the same interface, either by using the Horizon dashboard or using the command-line interface. Stacks standardize and speed up delivery by providing a unified human-readable format. The orchestration project started as an analog of AWS CloudFormation, making it compatible with the template formats used by CloudFormation (CFN), but it also supports its native template format, *Heat Orchestration Template (HOT).* The orchestration service executes Heat Orchestration Template (HOT) written in YAML. The YAML format is a human-readable data serialization language.

The template, along with the input parameters, calls the orchestration REST APIs for deploying the stack using either the dashboard or the OpenStack CLI commands. The orchestration API service forwards requests to the orchestration engine service using remote procedure calls (RPCs) over AMQP. Optionally, the orchestration CFN service sends the AWS CloudFormation-compatible requests to the orchestration engine service over RPC. The orchestration engine service interprets the orchestration template and launches the stack. The events generated by the orchestration engine service are consumed by the orchestration API service to provide the status of the orchestration stack that was launched.

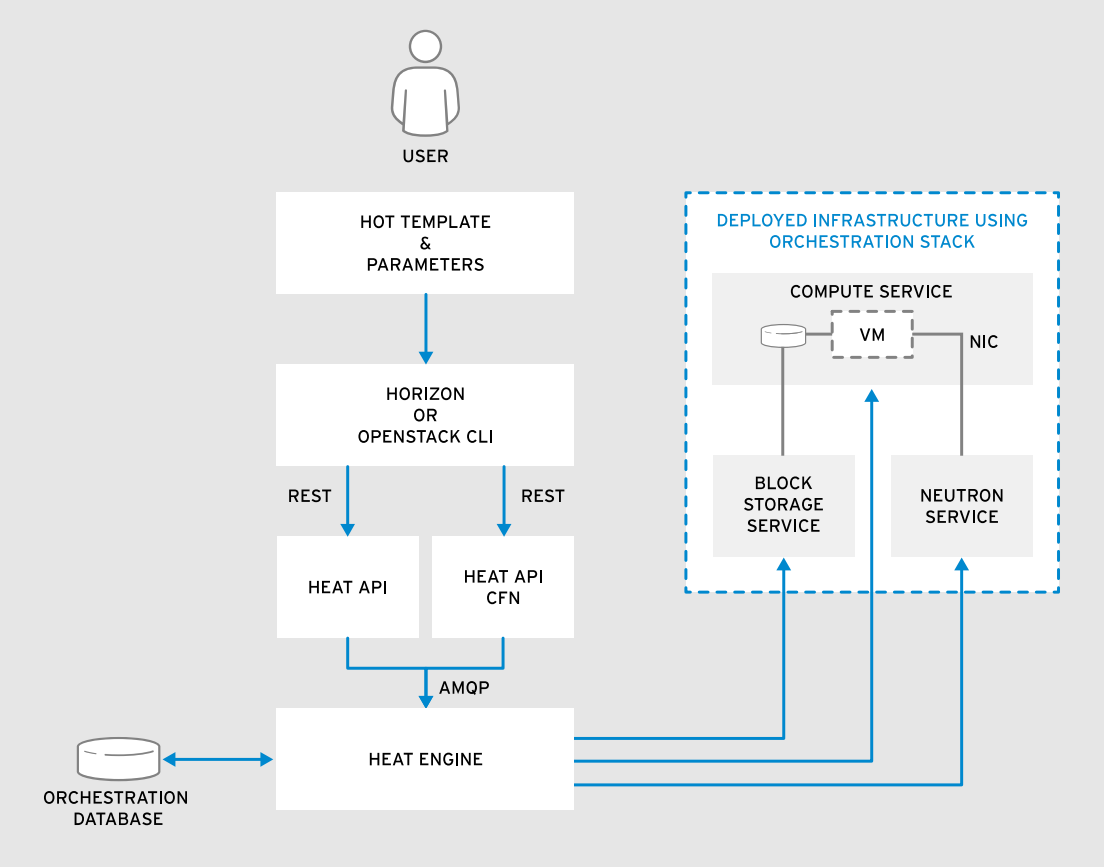


Figure 8.1: The Orchestration service architecture

The Heat Orchestration Service

In Red Hat OpenStack Platform, the orchestration service is provided by heat. An orchestration template specifies the relationship between the resources to be deployed. The relationship specified in the template enables the orchestration engine to call different OpenStack APIs to deploy the resources in the correct order. The orchestration template uses resource types to create various resources such as instances, volumes, security groups, and other resources. Next, more complex resources are created using a nested stack. The orchestration templates primarily deploy various infrastructural components. Different software configuration management tools, such as Ansible, Puppet, and others, can be integrated with the orchestration templates to deploy software and to make configuration changes to this software.

ORCHESTRATION USE CASES AND RECOMMENDED PRACTICES

The orchestration template can be used repeatedly to create identical copies of the same orchestration stack. A template, written in YAML format, can be placed under a source control system to maintain various versions of the infrastructure deployment. Orchestration makes it easy to organize and deploy a collection of OpenStack resources. An orchestration template allows you to describe any dependencies or pass specific parameters at run time. In an orchestration template, parameters are used to customize aspects of the template at run time during the creation of the stack. The following recommended practices can help you plan and organize the deployment of your orchestration stack.

- Using multiple layers of stacks that build on top of one another is the best way to organize an orchestration stack. Putting all the resources in one stack becomes cumbersome to manage when the stack is scaled, and broadens the scope of resources to be provisioned.

- When using nested stacks, resource names or IDs can be hard coded into the calling stack. However, hard coding resource names or IDs can make templates difficult to reuse, and may increase overhead to get the stack deployed.

- The changes in the infrastructure after updating a stack should be verified first by performing a dry run of the stack deployment.

- Before launching a stack, ensure all the resources to be deployed by the orchestration stack are within the project quota limits.

- With the growth of infrastructure, declaring resources in each template becomes repetitive. Such shared resources should be maintained as a separate stack and used inside a nested stack. Nested stacks are stacks that create other stacks.

- When declaring parameters in the orchestration template, use constraints to define the format for the input parameters. Constraints allow you to describe valid input values so that the orchestration engine catches any invalid values before creating the stack.

- Before using a template to create or update a stack, you can use the OpenStack CLI to validate it. Validating a template helps catch syntax and some semantic errors, such as circular dependencies before the orchestration stack creates any resources.

CONFIGURING ORCHESTRATION DOMAIN USERS

The orchestration stack domain users allow the orchestration service to authorize and perform operations inside the launched instances. The orchestration service APIs provide the metadata required to perform these operations. The following operations are performed inside an instance:

- The orchestration service provides metadata to the agents running inside the instance. These agents poll for the changes and apply the configuration that is provided by the orchestration API in the metadata to the instance.
- A signal is passed to the orchestration engine when an event is completed successfully or with the failed status. For example, when a software configuration on an instance is applied using cloud-init, a signal is passed to the orchestration service indicating the completion status.
- When an application provides application level status or meters from inside the instance to the orchestration engine. For example, to allow autoscaling actions to be performed in response to some measure of performance or quality of service.

A default overcloud deployment contains a `heat_stack` domain and a `heat_stack_domain_admin` user as the stack domain user. The `heat_stack_domain_admin` user is responsible for administering stack domain users and the administrative scope is limited to the `heat_stack` domain.

```
[user@demo ~]$ openstack user list --domain heat_stack
+----------------------------------+------------------------+
| ID                               | Name                   |
+----------------------------------+------------------------+
| e16340592b274fa2ae971aee988fb8e5 | heat_stack_domain_admin |
+----------------------------------+------------------------+
```

When a stack is deployed that requires creation of stack domain user to perform tasks inside the launched instance the orchestration service creates a new project within the `heat_stack` domain. The project within the stack domain is associated with the orchestration stack and is separate from the stack owner's project. For any resources within the orchestration stack that require a user, the orchestration service creates the user in the project within the stack domain. The users created are assigned the `heat_stack_user` role.

CONFIGURATION FILES AND LOGS

In the containerized architecture, the configuration files are stored in **`/var/lib/config-data/puppet-generated/heat/etc/heat`**. The orchestration service uses the **`heat.conf`** file for configuration. Some of the most common configuration options can be found in the following table:

Configuration Parameters for the Orchestration Service

PARAMETER	DESCRIPTION
`encrypt_parameters_and_properties`	Encrypts the parameter and properties of a resource. Marked as hidden before storing in the database. This parameter accepts a Boolean value.
`stack_user_domain_name`	The Identity domain name that contains orchestration service defined users. The parameter takes a string value. The default value is the **`heat_stack`** domain.
`stack_domain_admin`	The identity user name that is responsible for managing users and projects within the stack domain. This parameter takes a string value. The default value is the **`heat_stack_domain_admin`** user.
`heat_stack_user_role`	The identity user role name associated with the user created within the stack domain. This parameter takes a string value. The default value is the **`heat_stack_user`** role.
`num_engine_workers`	The number of `heat-engine` processes to fork and run on the host. This parameter takes an integer value. The default value is either the number of CPUs on the host running the `heat-engine` service, or 4, whichever is greater.
`stack_action_timeout`	The default timeout period in seconds to time out the creation and update of a stack. The default value is 3600 seconds (1 hour).

The log files for the orchestration service are stored in the **`/var/log/containers/heat`** directory of the host on which the `heat-api`, `heat-engine`, and `heat-api-cfn` services are running.

Log Files for the Orchestration Service

FILE NAME	DESCRIPTION
`heat-api.log`	Stores the log related to the orchestration API service.
`heat-api-cnf.log`	Stores the log related to the orchestration CloudFormation API service.
`heat-engine.log`	Stores the log related to the orchestration engine service.

TROUBLESHOOTING THE ORCHESTRATION SERVICE

Most errors occur while deploying the orchestration stack. The following are some of the more common errors and ways to troubleshoot the problem.

- Editing an existing template might introduce YAML syntax errors. Various tools, such as **`python -m json.tool`**, help locate the errors in the template files. Using the **`--dry-run`** option with the **`openstack stack create`** command validates some of the YAML syntax.

- If an instance goes into the **ERROR** state after launching a stack, troubleshoot the problem by looking for the **`/var/log/nova/scheduler.log`** log file on the compute node. If the error shows **`No valid host was found`**, the compute node does not have the required resources to launch the instance.

  Check the resources consumed by the instances running on the compute nodes. Some resource allocation issues can be mitigated by changing the overcommit allocation ratios in the **`/etc/nova/nova.conf`** file. By overcommitting, a higher number of instances can be deployed, but with the possibility of greater resource contention and lower performance.

  - CPU allocation ratio: 16:1. The scheduler allocates up to 16 virtual cores per physical core. For example, on a system with 12 cores, the Nova scheduler would see 192 available virtual cores. With a flavor definition that would configure 4 virtual cores per instance, the physical system could deploy 48 instances.

  - RAM allocation ratio: 1.5:1. The scheduler will continue to allocate instances if the configured memory of instances is up to 1.5 times the actual physical system RAM. For example, on a system with 48 GB of RAM, the Nova scheduler will continue to allocate instances until the total RAM deployed is 72 GB. If each instance was configured as 8 GB, nine instances could be built.

  To overcommit the amount of CPU and RAM allocated on the compute nodes, use the following commands to change the allocation ratio. The ratios shown here are the default values.

```
[user@demo ~]$ crudini --set /etc/nova/nova.conf DEFAULT cpu_allocation_ratio 16.0
[user@demo ~]$ crudini --set /etc/nova/nova.conf DEFAULT ram_allocation_ratio 1.5
```

- Use the **`--dry-run`** option when validating a template. It verifies the existence of resources required for the template and runtime parameters. Using custom constraints helps the template parameters to be parsed at an early stage rather than failing during the launch of the stack.

REFERENCES

For more information, refer to the *Components* chapter in the *Architecture Guide* at https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/12/html-single/architecture_guide/components#comp-orchestration

CREATING APPLICATION STACKS

Choose the correct answers to the following questions:

1. Which OpenStack service provides orchestration functionality in Red Hat OpenStack Platform?

a. Nova
b. Glance
c. Heat
d. Ceilometer

2. Which two template formats are supported by the orchestration service? (Choose two.)

a. OpenStack Orchestration Template (OOT)
b. Heat Orchestration Template (HOT)
c. Rapid Deployment Template (RDP)
d. CloudFormation (CFN)

3. In what language are orchestration templates written?

a. XML
b. JSON
c. YAML
d. HTML

4. What is the default timeout period for stack creation?

a. 86400 Seconds
b. 3600 Seconds
c. 300 Seconds
d. 600 Seconds

5. Which command-line interface option helps to validate a template?

a. `--validate`
b. `--run-dry`
c. `--dry-run`
d. `--yaml`

CREATING APPLICATION STACKS

Choose the correct answers to the following questions:

1. Which OpenStack service provides orchestration functionality in Red Hat OpenStack Platform?

a. Nova
b. Glance
c. Heat
d. Ceilometer

2. Which two template formats are supported by the orchestration service? (Choose two.)

a. OpenStack Orchestration Template (OOT)
b. Heat Orchestration Template (HOT)
c. Rapid Deployment Template (RDP)
d. CloudFormation (CFN)

3. In what language are orchestration templates written?

a. XML
b. JSON
c. YAML
d. HTML

4. What is the default timeout period for stack creation?

a. 86400 Seconds
b. 3600 Seconds
c. 300 Seconds
d. 600 Seconds

5. Which command-line interface option helps to validate a template?

a. `--validate`
b. `--run-dry`
c. `--dry-run`
d. `--yaml`

WRITING HEAT ORCHESTRATION TEMPLATES

OBJECTIVES

After completing this section, students should be able to write templates using the Heat Orchestration Template (HOT) language.

INTRODUCTION TO YAML

Orchestration templates are written using the *YAML Ain't Markup Language (YAML)* language. Therefore, it is necessary to understand the basics of YAML syntax to write an orchestration template.

YAML was designed primarily for the representation of data structures, such as lists and associative arrays, in an easily written, human-readable format. This design objective is accomplished primarily by abandoning traditional enclosure syntax, such as brackets, braces, or opening and closing tags, commonly used by other languages to denote the structure of a data hierarchy. Instead, in YAML, data hierarchy structures are maintained using outline indentation.

Data structures are represented using an outline format with space characters for indentation. There is no strict requirement regarding the number of space characters used for indentation other than data elements must be further indented than their parents to indicate nested relationships. Data elements at the same level in the data hierarchy must have the same indentation. You can use blank lines to help improve readability.

Indentation can only be performed using the space character. Indentation is critical to the correct interpretation of YAML. Because tabs are treated differently by various editors and tools, YAML does not support the use of tabs for indentation.

Add the following line to your **`~/.vimrc`** file, the **`Tab`** key is engages wisps indentation. This also automatically indents subsequent lines.

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

HEAT ORCHESTRATION TEMPLATE (HOT) LANGUAGE

Heat Orchestration Template (HOT) is a language supported by the orchestration service. The template uses the YAML syntax to describe various resources and properties.

Each orchestration template must include the **`heat_template_version`** key with a correct orchestration template version. The orchestration template version defines both the supported format of the template and the features that are valid and supported for the orchestration service. The **`openstack orchestration template version list`** command lists all the supported template versions. Names listed in the **`Aliases`** column of the command output can also be used specify the template version.

```
[user@demo ~]$ openstack orchestration template version list
+-------------------------------------+------+------------------------------+
| Version                             | Type | Aliases                      |
+-------------------------------------+------+------------------------------+
| AWSTemplateFormatVersion.2010-09-09 | cfn  |                              |
| HeatTemplateFormatVersion.2012-12-12 | cfn  |                              |
```

```
| heat_template_version.2013-05-23     | hot  |                              |
| heat_template_version.2014-10-16     | hot  |                              |
| heat_template_version.2015-04-30     | hot  |                              |
| heat_template_version.2015-10-15     | hot  |                              |
| heat_template_version.2016-04-08     | hot  |                              |
| heat_template_version.2016-10-14     | hot  | heat_template_version.newton |
| heat_template_version.2017-02-24     | hot  | heat_template_version.ocata  |
| heat_template_version.2017-09-01     | hot  | heat_template_version.pike   |
| heat_template_version.2018-03-02     | hot  | heat_template_version.queens |
+--------------------------------------+------+------------------------------+
```

The **`description`** key in a template is optional, but can include text that describes the purpose of the template. You can add multiline text to the **`description`** key by using *folded blocks* (>) in YAML. Folded blocks replace each line break with a single space, ignoring indentation.

```
heat_template_version: queens
description: >
  This is multiline description
  that describes the template usage.
```

Customizing Templates with Input Parameters

You can use input parameters to customize the template during deployment of the orchestration stack. The input parameters are defined in the **`parameters`** section of the orchestration template. Each parameter is defined as a separate nested block with required properties such as **`type`** or **`default`**.

In the orchestration template, the **`parameters`** section uses the following syntax and attributes to define an input parameter for the template.

```
parameters:
  <param_name>:
    type: <string | number | json | comma_delimited_list | boolean>
    label: <human-readable name of the parameter>
    description: <description of the parameter>
    default: <default value for parameter>
    hidden: <true | false>
    constraints:
      <parameter constraints>
    immutable: <true | false>
```

The following table shows the list of supported attributes for defining the input parameters:

Table of Supported Attributes for Input Parameters

ATTRIBUTE	DESCRIPTION
`type`	The data type of the parameter. The supported data types are string, number, JSON, comma-delimited list, and Boolean.
`label`	A human-readable name for the parameter. This attribute is optional.
`description`	A short description of the parameter. This attribute is optional.

ATTRIBUTE	DESCRIPTION
default	The default value to use if no value for the parameter is entered. This attribute is optional.
hidden	Determines whether the value of the parameter is hidden in list information about a stack created by the orchestration template. This attribute is optional and defaults to false.
constraints	Constraints to be applied to validate the input value for a parameter. The constraints attribute can apply to lists of different constraints. This attribute is optional.
immutable	Defines whether the parameter can be updated. The stack fails to update if the parameter value is changed and the attribute value is set to **true**.

The **custom_constraints** attribute adds an extra step of validation to verify that the resource exists. Custom constraints are implemented using orchestration plug-ins. The **custom_constraints** attribute uses the name associated with the orchestration plug-ins. For example, use the following syntax to ensure the existence of a block storage (cinder) volume:

```
parameters:
  volume_name:
    type: string
    description: volume name
    constraints:
      - custom_constraints: cinder.volume
```

Defining Resources During Stack Deployment

The **resources** section in the orchestration template defines resources provisioned during deployment of a stack. Each resource is defined as a separate nested block with its required attributes, such as **type** and **properties**. The **properties** attribute defines the properties required to provision the resource.

The **resources** section in a template uses the following syntax and attributes to define a resource for the stack.

```
resources:
  <resource ID>:
    type: <resource type>
    properties:
      <property name>: <property value>
```

The following table shows the list of attributes for defining a resource:

Table of Supported Attributes for Resources

ATTRIBUTE	DESCRIPTION
resource ID	A user-defined resource name. This must be uniquely referenced within the **resources** section of the template.

ATTRIBUTE	DESCRIPTION
type	This attribute uses the resource type name. The core OpenStack resources are included in the orchestration engine service as a built-in resource. The orchestration service provides support for resource plug-ins using custom resources. This attribute is mandatory and must be specified when declaring a resource.
properties	This attribute is used to specify a list of properties associated with a resource type. The property value is either hard-coded or uses intrinsic functions to retrieve the value. This attribute is optional.

Resource Types

A resource requires a **type** property, such as an instance and various properties, that depend on the resource type. Use the **openstack orchestration resource type list** command to list the available resource types.

```
[user@demo ~]$ openstack orchestration resource type list
+---------------------------------------------+
| Resource Type                               |
+---------------------------------------------+
...output omitted...
| OS::Nova::FloatingIP                        |
| OS::Nova::FloatingIPAssociation             |
| OS::Nova::KeyPair                           |
| OS::Nova::Server                            |
| OS::Nova::ServerGroup                       |
| OS::Swift::Container                        |
+---------------------------------------------+
```

The **OS::Heat::ResourceGroup** resource type creates one or more identical resources. The resource definition is passed as a nested stack. The required property for the **ResourceGroup** resource type is **resource_def**. The value of the **resource_def** property is the definition of the resource to be provisioned. The **count** property sets the number of resources to provision. For example, use the following syntax to create two resources of the **OS::Nova::Server** resource type.

```
resources:
  my_group:
    type: OS::Heat::ResourceGroup
    properties:
      count: 2
      resource_def:
        type: OS::Nova::Server
        properties:
          name: { get_param: instance_name }
          image: { get_param: instance_image }
```

The **index_var** property sets a value that the resource in the resource group uses to replace with the current index of a given resource in the group, to customize the **name** property of grouped instances in order to differentiate them when listed. The default value set for the property is **%index%**, which provides the current index of a given resource in the resource group.

In this example, a resource group is defined to launch four instances. Use the **`list_join`** function and the **`%index%`** value to name these instances as follows:

- `demo0.overcloud.example.com`
- `demo1.overcloud.example.com`
- `demo2.overcloud.example.com`
- `demo3.overcloud.example.com`

```
resources:
  my_group:
    type: OS::Heat::ResourceGroup
    properties:
      count: 4
      resource_def:
        type: OS::Nova::Server
        properties:
          name:
            list_join: [ "%index%", [ "demo", ".overcloud.example.com" ] ]
          image: { get_param: instance_image }
```

Intrinsic Functions

HOT provides several built-in functions that are used to perform specific tasks in the template. Intrinsic functions in an orchestration template assign values to the properties that are available during creation of the stack. Some of the widely used intrinsic functions are listed below:

`get_attr`

The **`get_attr`** function references a property of a resource. This function takes the resource name and the property name as the parameters to retrieve the property value for the resource. The example shows how the **`get_attr`** function retrieves the first IP address associated with the `the_instance` resource that is set as an output value of the stack.

```
resources:
  the_instance:
    type: OS::Nova::Server
    ...output omitted...

outputs:
  instance_ip:
  description: IP address of the instance
  value: { get_attr: [the_instance, first_address] }
```

`get_param`

The **`get_param`** function references an input parameter of a template and returns the value of the input parameter. This function takes the parameter name as the parameter to retrieve the value of the input parameter declared in the template. The example shows how the **`get_param`** function retrieves the value of the **`instance_flavor`** input parameter that is set as a value for the **`flavor`** property.

```
parameters:
  instance_flavor:
    type: string
    description: Flavor to be used by the instance.

resources:
  the_instance:
```

```
    type: OS::Nova::Server
    properties:
      flavor: { get_param: instance_flavor }
```

get_resource

The **get_resource** function references a resource in the template. The function takes the resource name as the parameter to retrieve the resource ID of the referenced resource. The example shows how the **get_resource** function retrieves the resource ID of the `the_port` resource that is set as a value for the **port** property.

```
resources:
  the_port:
    type: OS::Neutron::Port
      ...output omitted...

    the_instance:
      type: OS::Nova::Server
      properties:
        networks:
          port: { get_resource: the_port }
```

str_replace

The **str_replace** function substitutes variables in an input string with values that you specify. The input string along with variables are passed to the **template** property of the function. The values of the variables are instantiated using the **params** property as a key-value pair. The example shows how the **str_replace** function substitutes the `varname` variable in the template with the first IP address of the `the_instance` resource.

```
outputs:
  website_url:
    description: The website URL of the application.
    value:
      str_replace:
        template: http://varname/MyApps
        params:
          varname: { getattr: [ the_instance, first_address ] }
```

list_join

The **list_join** function appends a set of strings into a single value, separated by the specified delimiter. If the delimiter is an empty string, it concatenates all of the strings. The example shows how the **list_join** function concatenates the instance name with a random value that is used to set the value for the **instance_name** property.

```
resources:
  random:
    type: OS::Heat::RandomString
    properties:
      length: 2

  the_instance:
    type: OS::Nova::Server
    properties:
      instance_name: { list_join: [ '-', [ {get_param: instance_name}, {get_attr:
 [random, value]} ] ] }
```

CREATING WAIT CONDITIONS IN A TEMPLATE

You can use the wait condition and wait condition handle to pause the creation of a stack and wait for a signal before it continues to deploy the stack. For example, you might want to download and configure applications on an instance before considering the creation of that instance complete. The following list provides a summary of how a wait condition with a wait condition handle works:

- When an orchestration template creates a wait condition and references the wait condition handle to a resource, it reports the wait condition's status as **`CREATE_IN_PROGRESS`**. The stack deployment waits until it receives the requisite number of success signals or the wait condition's timeout period has expired. The requisite number of success signals is defined by the **`count`** property of the **`OS::Heat::WaitCondition`** resource type. The timeout is defined by the **`timeout`** property of the **`OS::Heat::WaitCondition`** resource type.
- If the orchestration service receives the requisite number of success signals before the timeout period expires, it continues creating the stack; otherwise, it sets the wait condition's status to **`CREATE_FAILED`**.
- If a wait condition receives the requisite number of success signals before the timeout period expires, the orchestration service marks the wait condition as **`CREATE_COMPLETE`** and continues creating the stack.

To use a wait condition in an orchestration template, declare the following resources:

1. Declare a resource of the **`OS::Heat::WaitConditionHandle`** resource type in the stack's template. A wait condition handle resource has no properties. However, a reference to a wait condition handle resource resolves to a presigned URL that the resource can use to signal success or failure to the wait condition. The following example declares a wait condition handle resource named `demo_wait_handle` in a template:

```
demo_wait_handle:
  type: OS::Heat::WaitConditionHandle
```

   The presigned URL that resolves to the wait condition handle contains the token that is generated using credentials for a user account with limited access to the wait condition handle resource. This user is created when the wait condition handle is created, and is associated with a project that belongs to the stack, in an identity domain that is dedicated to the orchestration service.
2. Declare an **`OS::Heat::WaitCondition`** resource type in the stack's template. A wait condition resource has two required properties: **`handle`**, which is a reference to a wait condition handle declared in the template; and **`timeout`**, which is the number of seconds for the orchestration service to wait before resuming creation of the stack. You can optionally set the **`count`** property, which determines the number of success signals that the wait condition must receive before the orchestration service can resume creating the stack. The following example declares a wait condition resource named `demo_wait_condition`. The resource uses `demo_wait_handle` as the wait condition handle, has a timeout of 600 seconds, and has a default count of 2.

```
demo_wait_condtion:
  type: OS::Heat::WaitCondition
  properties:
    handle: { get_resource: demo_wait_handle }
    count: 2
    timeout: 600
```

3. The defined wait condition handle resource is referenced to receive the signals to mark the creation of a resource as **`CREATE_COMPLETE`** or **`CREATE_FAILED`**. The following template example uses the **`curl_cli`** attribute for notification. In the example below, the

demo_server resource is marked as **CREATE_COMPLETE** after it receives two success signals:

```
demo_server:
  type: OS::Nova::Server
  properties:
    ...output omitted...
    user_data:
      str_replace:
        template: |
          #!/bin/bash
          echo $(hostname) > /tmp/test1
          [[ -f /tmp/test1 ]] && $demo_notify --data-binary \
          '{"status": "SUCCESS", "reason": "signal1"}'
          echo $(whoami) > /tmp/test2
          [[ -f /tmp/test2 ]] && $demo_notify --data-binary \
          '{"status": "SUCCESS", "reason": "signal2"}'
        params:
          demo_notify: { get_attr: [demo_wait_handle, curl_cli] }
```

SOFTWARE CONFIGURATION USING HEAT ORCHESTRATION TEMPLATES

Orchestration templates provide a variety of options to configure software on the instance provisioned by the orchestration stack. The frequency of the software configuration changes to be applied to the software installed on the instance is the deciding factor on how to implement software configuration. There are, broadly, three options for implementing software configuration changes using the orchestration template:

- Use a custom image that includes installed and configured software. This method can be used when there is no change in software configuration required during the life cycle of an instance.
- Use the user-data script and cloud-init to configure the preinstalled software in the image. This method can be used when there is a software configuration change required once during the life cycle of an instance (at boot time). An instance must be replaced with a new instance when software configuration changes are made using this option.
- Use the **OS::Heat::SoftwareDeployment** resource type. This allows any number of software configuration changes to be applied to an instance throughout its life cycle.

Specifying User-data Scripts in a Heat Orchestration Template

When provisioning an instance, you can specify a user-data script to configure the software installed on the instance. Software can be installed into the image, or installed using a user-data script. In the HOT language, user data is provided using the **user-data** property of the **OS::Nova::Server** resource type. The data provided using the **user-data** property can be a shell script or a cloud-init script. The **str_replace** intrinsic function is used to set the variable value based on the parameters or the resources in a stack. The **user_data_format** property defines the way user data is processed by an instance. Using **RAW** as the value of the **user_data_format** property, the user data is passed to the instance unmodified.

In the example below, the user data is defined for a resource using the **str_replace** function. The instance is launched and the user-data script is executed to create a file named **/tmp/demofile**. The $demo variable is set to use **demofile** as the name of the file.

```
resources:
  the_instance:
```

```
    type: OS::Nova::Server
    properties:
        ...output omitted...
      user_data_format: RAW
      user_data:
        str_replace:
          template: |
            #/bin/bash
            echo "Hello World" > /tmp/$demo
          params:
            $demo: demofile
```

When the user data is changed and the orchestration stack is updated using the **`openstack stack update`** command, the instance is deleted and recreated using the updated user-data script.

To provide complex scripts using the **`user-data`** property, use the **`get_file`** intrinsic function. The **`get_file`** function takes the name of a file as its argument.

In the example below, the **`get_file`** function takes the name of the script file as an argument to execute as a user-data script.

```
resources:
  the_instance:
    type: OS::Nova::Server
    properties:
      ...output omitted...
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: demoscript.sh }
          params:
            $demo: demofile
```

Describing the Software Deployment Resource Type

Use the **`OS::Heat::SoftwareDeployment`** resource type to initiate software configuration changes without replacing the instance with a new instance. An example use case is any situation where an instance cannot be replaced with a new instance, but software configuration changes are needed during the life cycle of the instance. The **`OS::Heat::SoftwareDeployment`** resource type allows you to add or remove software configuration multiple times from an instance during its life cycle. There are three resource types required to perform the software configuration changes using the orchestration stack.

- The **`OS::Heat::SoftwareConfig`** resource type enables integration with various software configuration tools such an Ansible, a shell script, or Puppet. The resource type creates an immutable software configuration so that any change to software configuration replaces the old configuration with a new configuration. Properties of the **`OS::Heat::SoftwareConfig`** resource type are **`config`**, **`group`**, **`inputs`**, and **`outputs`**. The **`group`** property defines the name of the software configuration tool to use, such as a script, Ansible, or Puppet. The **`config`** property sets the configuration script or manifest that specify the actual software configuration performed on the instance. The **`inputs`** and **`outputs`** properties represent the input and output options for the software configuration.

  In the following example the software configuration resource the **`group`** property is set to **`script`**. The input parameters are set to use the **`filename`**, and **`content`** items defined for

the **input** property. The output is returned using the **result** item defined for the **outputs** property. The **config** property sets the configuration script to use **demo-script.sh**.

```
resources:
  the_config:
    type: OS::Heat::SoftwareConfig
    properties:
      group: script
      inputs:
        - name: filename
        - name: content
      outputs:
        - name: result
      config: { get_file: demo-script.sh }
```

- The **OS::Heat::SoftwareDeployment** resource type applies the software configuration defined using the **OS::Heat::SoftwareConfig** resource type. The **SoftwareDeployment** resource type allows you to input values based on defined input variables using the **inputs** property of the **SoftwareConfig** resource type. When the state changes to **IN_PROGRESS**, the software configuration that has been replaced with the variable values is made available to the instance. The state is changed to **CREATE_COMPLETE** when a success signal is received from the orchestration API.

  The required property for the **OS::Heat::SoftwareDeployment** resource type is the **server** property. The **server** property is a reference to the ID of the resource to which configuration changes are applied. Other optional properties includes **action**, **config**, and **input_values**. The **action** property defines when the software configuration needs to be initiated based on the orchestration stack state. The **action** property supports **CREATE**, **UPDATE**, **SUSPEND**, and **RESUME** actions. The **config** property references the resource ID of the software configuration resource to execute when applying changes to the instance. The **input_values** property maps values to the input variables defined in the software configuration resource.

  In the following example using the **input_values** property, the input parameters are passed to the software configuration resource, `the_config`. The software deployment is invoked when creating or updating the stack to either create or update the `the_server` resource.

```
resources:
  the_deployment:
    type: OS::Heat::SoftwareDeployment
    properties:
      server: { get_resource: the_server }
      actions:
        - CREATE
        - UPDATE
      config: { get_resource: the_config }
      input_values:
        filename: demofile
        content: 'Hello World'
```

- The **OS::Nova::Server** resource type defines the instance on which the software configuration changes are applied. The **user_data_format** property of the **OS::Nova::Server** resource type must use the **SOFTWARE_CONFIG** value to support the software configuration changes using the **OS::Heat::SoftwareDeployment** resource type.

```
resources:
  the_server:
    type: OS::Nova::Server
    properties:
        ...output omitted...
      user_data_format: SOFTWARE_CONFIG
```

The **`OS::Heat::SoftwareDeployment`** resource type requires orchestration agents to collect and process the configuration changes by polling the orchestration API. You must embed these orchestration agents into the image. The *python-heat-agent* package must be included, and provides support for software configuration via shell scripts. Support for other software configuration tools is available from the *python-heat-agent-ansible* package (for Ansible Playbooks) or the *python-heat-agent-puppet* package (for Puppet manifests).

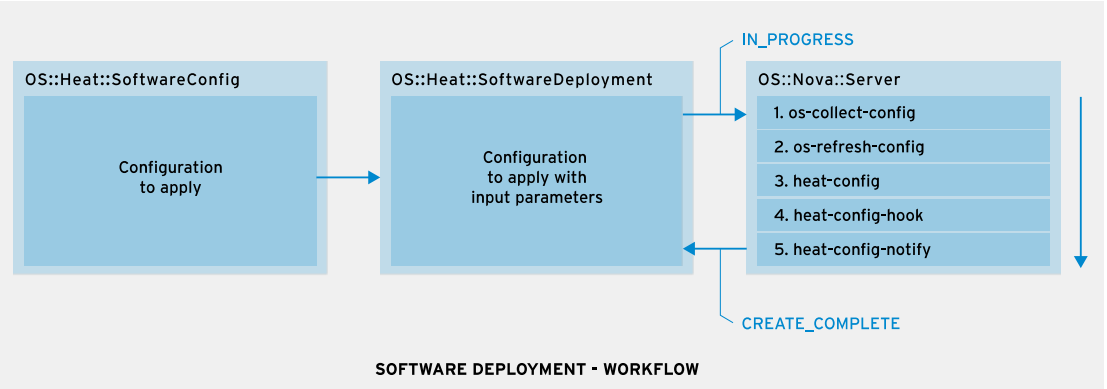


Figure 8.2: Software deployment workflow

Other agents used to apply software configuration changes on an instance are shown below:

- The **`os-collect-config`** agent poll the orchestration API for updated resource metadata that is associated with the **`OS::Nova::Server`** resource type.
- The **`os-refresh-config`** agent is executed when there is a change in the software configuration and which is polled by the **`os-collect-config`** agent. It refreshes the configuration by deleting the older configuration and replacing it with the newer configuration. The **`os-refresh-config`** agent uses the **`group`** property defined for the deployment to process the configuration. It uses the **`heat-config-hook`** scripts to apply the software configuration changes. The **`heat-config-hook`** scripts are provided by the *python-heat-agent-***** packages. Upon completion, the hook notifies the orchestration API of a successful or failed configuration deployment using the **`heat-config-notify`** script.
- The **`os-apply-config`** agent transforms software configuration data provided by the orchestration template into a service configuration file.

REFERENCES

Template Guide
https://docs.openstack.org/heat/latest/template_guide/index.html

Software configuration
https://docs.openstack.org/heat/latest/template_guide/software_deployment.html

WRITING HEAT ORCHESTRATION TEMPLATES

In this exercise, you will edit an orchestration template to launch a multitier orchestration stack.

OUTCOMES

You should be able to:

- Edit an orchestration template to launch a multitier stack.
- Launch a stack using the orchestration template.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab orchestration-heat-templates setup`** command. This script ensures that the OpenStack services are running and the environment is properly configured for the exercise. The script also ensures that the resources needed for launching the stack are available and downloads the required scripts to the **`/home/student/heat-templates`** directory.

```
[student@workstation ~]$ lab orchestration-heat-templates setup
```

1. When you edit YAML files, you must use spaces, not tab characters, for indentation. If you use **`vi`** for text editing, configure the **`.vimrc`** file to set auto-indentation and set the `ts` and `sw` variables to two spaces for YAML files. Edit the **`/home/student/.vimrc`** file with the following content:

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

2. Download the `http://materials.example.com/heat/finance-app1.yaml` file to the **`/home/student/heat-templates`** directory. Edit the orchestration template to launch a customized instance.

   The orchestration template must include the following items:

   - The `finance-web` instance must deploy a PHP web page that interacts with the MariaDB database. Run the **`webserver.sh`** script as the user-data script to deploy a PHP web page on the `finance-web` instance.
   - The `finance-db` instance must configure a MariaDB database. Run the **`dbserver.sh`** script as the user-data script to deploy a MariaDB database and to create a database named `mydb`.
   - The following parameters must be passed to the **`webserver.sh`** script using the **`user_data`** property:

The `$web_public_ip` and `$web_private_ip` variables must be initialized with the external and private IP addresses of the `finance-web` instance, respectively. The `$db_private_ip` variable must be initialized with the private IP address of the `finance-db` instance. These variables are used in the **`webserver.sh`** script.

- The orchestration stack must retry once to execute the user-data script. The user-data script must return success on the successful execution of the script. Set the timeout period to 600 seconds, after which the **`webserver.sh`** script must return a **`fail`** result. Set the timeout period to 800 seconds, after which the **`dbserver.sh`** script must return a **`fail`** result.

2.1. Change to the **`/home/student/heat-templates`** directory. Download the `http://materials.example.com/heat/finance-app1.yaml` orchestration template file to the **`/home/student/heat-templates`** directory.

```
[student@workstation ~]$ cd ~/heat-templates
[student@workstation heat-templates]$ wget \
http://materials.example.com/heat/finance-app1.yaml
```

2.2. Use the **`user_data`** property to define the **`/home/student/heat-templates/webserver.sh`** user-data script to deploy a PHP web page to `finance-web`. Set the **`user_data_format`** property of the **`OS::Nova::Server`** resource type to **`RAW`**.

Edit the **`/home/student/heat-templates/finance-app1.yaml`** file and add the content highlighted below:

```
  web_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: web_instance_name }
      image: { get_param: web_image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: web_net_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: /home/student/heat-templates/webserver.sh }
```

2.3. The web page uses the `$web_private_ip`, `$db_private_ip`, and `$web_public_ip` variables passed as parameters. These parameters are defined using the **`params`** property of the **`str_replace`** intrinsic function.

The `$web_private_ip` variable uses the `web_net_port` resource attribute, **`fixed_ips`**, to retrieve the first IP address associated with the `web_server` resource. The `$db_private_ip` variable uses the `db_net_port` resource attribute, **`fixed_ips`**, to retrieve the first IP address associated with the `db_server` resource. The `$web_public_ip` variable uses the `web_floating_ip` resource attribute, **`floating_ip_address`**, to set the public IP address associated with the `web_server` resource.

Edit the **`/home/student/heat-templates/finance-app1.yaml`** file and add the content highlighted below:

```
  web_server:
```

```
    type: OS::Nova::Server
    properties:
      name: { get_param: instance_name }
      image: { get_param: image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: web_net_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: /home/student/heat-templates/webserver.sh }
          params:
            $web_private_ip: {get_attr: [web_net_port,fixed_ips,0,ip_address]}
            $db_private_ip: {get_attr: [db_net_port,fixed_ips,0,ip_address]}
            $web_public_ip: {get_attr: [web_floating_ip,floating_ip_address]}
```

2.4. Use the **`WaitHandleCondition`** resource to send a signal about the status of the user-data script.

The `$wc_notify` variable is set to the wait handle URL using the **`curl_cli`** attribute of the `web_wait_handle` resource. The wait handle URL value is set to the `$wc_notify` variable. If the web page deployed by the script is accessible (HTTP status code 200), then the `$wc_notify` variable returns a status of **SUCCESS**. The `web_server` resource state is marked as **CREATE_COMPLETE** when the **`WaitConditionHandle`** resource signals **SUCCESS**.

The **`WaitConditionHandle`** returns **FAILURE** if the web page is not accessible or if it times out after 600 seconds. The `web_server` resource state is then marked as **CREATE_FAILED**.

Edit the **`/home/student/heat-templates/finance-app1.yaml`** file and add the content highlighted below:

```
  web_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: instance_name }
      image: { get_param: image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: web_net_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: /home/student/heat-templates/webserver.sh }
          params:
            $web_private_ip: {get_attr: [web_net_port,fixed_ips,0,ip_address]}
            $db_private_ip: {get_attr: [db_net_port,fixed_ips,0,ip_address]}
            $web_public_ip: {get_attr: [web_floating_ip,floating_ip_address]}
            $wc_notify: {get_attr: [web_wait_handle,curl_cli]}
```

2.5. Use the **`user_data`** property to define the **`/home/student/heat-templates/dbserver.sh`** user-data script to configure the MariaDB server on `finance-db`.

Set the **`user_data_format`** property of the **`OS::Nova::Server`** resource type to **`RAW`**.

Use the **`WaitHandleCondition`** resource to send a signal about the status of the user-data script. The `$wc_notify` variable is set to the wait handle URL using the **`curl_cli`** attribute of the `db_wait_handle` resource.

Edit the **`/home/student/heat-templates/finance-app1.yaml`** file and add the content highlighted below:

```
  db_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: db_instance_name }
      image: { get_param: db_image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: web_net_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: /home/student/heat-templates/dbserver.sh }
          params:
            $db_wc_notify: {get_attr: [db_wait_handle,curl_cli]}
```

Save and exit the file.

3. Create the **`/home/student/heat-templates/environment.yaml`** environment file. Enter the values for all input parameters defined in the **`/home/student/heat-templates/finance-app1.yaml`** template file.

   Create the **`/home/student/heat-templates/environment.yaml`** file with the following content:

```
parameters:
  web_image_name: rhel7-web
  db_image_name: rhel7-db
  web_instance_name: finance-web
  db_instance_name: finance-db
  instance_flavor: default
  key_name: example-keypair
  public_net: provider-datacentre
  private_net: finance-network1
  private_subnet: finance-subnet1
```

4. Launch the stack and verify it by accessing the web page deployed on the instance. Use the `developer1` user credentials to launch the stack.

4.1. Use the `developer1` user credentials to perform a dry-run stack deployment and inspect the resources that will be created when you launch the stack. Rectify all errors before proceeding to the next step.

Use the **`finance-app1.yaml`** template file and the **`environment.yaml`** environment file to create the stack. Name the stack `finance-app1`.

NOTE

Before you perform a dry run of deploying the stack, download the `http://materials.example.com/heat/finance-app1.yaml-final` template file to the **`/home/student/heat-templates`** directory. Use the **`diff`** command to compare the **`finance-app1.yaml`** template file with the known good template file, **`finance-app1.yaml-final`**. Fix any differences you find, and then proceed to launch the stack.

```
[student@workstation heat-templates]$ wget \
http://materials.example.com/heat/finance-app1.yaml-final
[student@workstation heat-templates]$ diff finance-app1.yaml \
finance-app1.yaml-final
```

```
[student@workstation heat-templates]$ source ~/developer1-finance-rc
[student@workstation heat-templates(developer1-finance)]$ openstack stack \
create \
--environment environment.yaml \
--template finance-app1.yaml \
--dry-run -c description \
finance-app1
+-------------+--------------------------+
| Field       | Value                    |
+-------------+--------------------------+
| description | multi-tier stack template |
+-------------+--------------------------+
```

4.2. Launch the stack using the **`finance-app1.yaml`** template file and the **`environment.yaml`** environment file. Name the stack `finance-app1`.

If the dry run is successful, use the **`openstack stack create`** command to launch the stack. Do not use the **`--dry-run`** option while launching the stack.

```
[student@workstation heat-templates(developer1-finance)]$ openstack stack \
create \
--environment environment.yaml \
--template finance-app1.yaml \
--wait \
finance-app1
...output omitted...
```

4.3. List the output returned by the `finance-app1` stack.

```
[student@workstation heat-templates(developer1-finance)]$ openstack stack \
output list finance-app1
```

```
+----------------+--------------------------------------------------+
| output_key     | description                                      |
+----------------+--------------------------------------------------+
| web_private_ip | Private IP address of the web server             |
| web_public_ip  | External IP address of the web server            |
| website_url    | This URL is the "external" URL that can be used  |
|                | to access the web server.                        |
|                |                                                  |
| db_private_ip  | Private IP address of the DB server              |
+----------------+--------------------------------------------------+
```

4.4. Inspect the **website_url** output value of the stack. The IP address in the **output_value** field might vary.

```
[student@workstation heat-templates(developer1-finance)]$ openstack stack \
output show finance-app1 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://172.25.250.107/index.php           |
+--------------+-------------------------------------------+
```

4.5. Verify that the multitier stack was created and the user-data script was executed successfully on both the instances by accessing the URL returned as the value for the **website_url** output.

Cleanup

On workstation, run the **lab orchestration-heat-templates cleanup** command to clean up this exercise.

```
[student@workstation ~]$ lab orchestration-heat-templates cleanup
```

This concludes the guided exercise.

AUTOMATING CLOUD APPLICATIONS

PERFORMANCE CHECKLIST

In this exercise, you will edit an orchestration template to launch identical customized web servers.

OUTCOMES

You should be able to:

- Edit an orchestration template to launch multiple customized web servers.
- Provision identical resources using the **`OS::Heat::ResourceGroup`** resource type.
- Launch a stack using the orchestration template.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab orchestration-review setup`** command. The script downloads the orchestration templates and environment file to the **`/home/student/heat-templates`** directory.

```
[studentworkstation ~]$ lab orchestration-review setup
```

1. Edit the **`/home/student/heat-templates/production-stack.yaml`** orchestration template. Add a new input parameter named **`instance_count`** under the **`parameters`** section. The parameter must define constraints to specify a minimum of one and a maximum of two instances.
2. Edit the **`/home/student/heat-templates/production-stack.yaml`** orchestration template. Add a resource named `production_resource` to launch identical resources based on user input. Use the **`OS::Heat::ResourceGroup`** resource type.
3. Edit the **`/home/student/heat-templates/production-environment.yaml`** environment file to set the **`instance_count`** input parameter to **`2`**.
4. Edit the **`/home/student/heat-templates/production-environment.yaml`** environment file to define a custom resource type named **`Production::Custom::WebServer`**. This custom resource type must point to the **`/home/student/heat-templates/production-app1.yaml`** template.
5. Edit the **`/home/student/heat-templates/production-stack.yaml`** orchestration template. Set the resource definition of the `production_resource` resource to use the **`Production::Custom::WebServer`** custom resource type. Set the required properties to launch the custom resource.

   Use the **`list_join`** intrinsic function and the **`%index%`** default value of the **`index_var`** property of **`OS::Heat::ResourceGroup`** to concatenate the instance name with the index of the resource group as the suffix. Set the **`instance_name`** property of the resource to use the concatenated name. Name the instances `production-server0` and `production-server1`.

6. Source the **/home/student/operator1-production-rc** identity environment file to launch the stack as `operator1`. Launch the stack using the **production-stack.yaml** template file and the **production-environment.yaml** environment file. Name the stack `production-webstack`.

7. Verify that the `production-webstack` stack provisioned two instances. Use the **instance_ipaddress** output parameter of the `production-webstack` stack to retrieve the private IP addresses attached to the instances.

Evaluation

On `workstation`, run the **lab orchestration-review grade** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~] lab orchestration-review grade
```

Cleanup

On `workstation`, run the **lab orchestration-review cleanup** command to clean up this exercise.

```
[student@workstation ~] lab orchestration-review cleanup
```

This concludes the lab.

AUTOMATING CLOUD APPLICATIONS

PERFORMANCE CHECKLIST

In this exercise, you will edit an orchestration template to launch identical customized web servers.

OUTCOMES

You should be able to:

- Edit an orchestration template to launch multiple customized web servers.
- Provision identical resources using the **`OS::Heat::ResourceGroup`** resource type.
- Launch a stack using the orchestration template.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab orchestration-review setup`** command. The script downloads the orchestration templates and environment file to the **`/home/student/heat-templates`** directory.

```
[studentworkstation ~]$ lab orchestration-review setup
```

1. Edit the **`/home/student/heat-templates/production-stack.yaml`** orchestration template. Add a new input parameter named **`instance_count`** under the **`parameters`** section. The parameter must define constraints to specify a minimum of one and a maximum of two instances.

   Use the **`range`** constraint to specify the minimum number of one and a maximum number of two instances.

```
parameters:
  ...output omitted...
  instance_count:
    type: number
    description: count of servers to be provisioned
    constraints:
      - range: { min: 1, max: 2 }
```

2. Edit the **`/home/student/heat-templates/production-stack.yaml`** orchestration template. Add a resource named `production_resource` to launch identical resources based on user input. Use the **`OS::Heat::ResourceGroup`** resource type.

   Set the **`count`** property to use the **`instance_count`** input parameter.

```
...output omitted...
resources:
```

```
  production_resource:
    type: OS::Heat::ResourceGroup
    properties:
      count: { get_param: instance_count }
      ...output omitted...
```

Save and exit the file.

3. Edit the **/home/student/heat-templates/production-environment.yaml** environment file to set the **instance_count** input parameter to **2**.

```
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"
  instance_count: 2
```

4. Edit the **/home/student/heat-templates/production-environment.yaml** environment file to define a custom resource type named **Production::Custom::WebServer**. This custom resource type must point to the **/home/student/heat-templates/production-app1.yaml** template.

```
resource_registry:
  Production::Custom::WebServer: /home/student/heat-templates/production-app1.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"
  instance_count: 2
```

Save and exit the file.

5. Edit the **/home/student/heat-templates/production-stack.yaml** orchestration template. Set the resource definition of the `production_resource` resource to use the **Production::Custom::WebServer** custom resource type. Set the required properties to launch the custom resource.

   Use the **list_join** intrinsic function and the **%index%** default value of the **index_var** property of **OS::Heat::ResourceGroup** to concatenate the instance name with the index of the resource group as the suffix. Set the **instance_name** property of the resource to use the concatenated name. Name the instances `production-server0` and `production-server1`.

   Set the **resource_def** property of the `production_resource` resource type to use the **Production::Custom::WebServer** custom resource type. The **Production::Custom::WebServer** custom resource type uses the input parameters required to provision the instance. Edit the file to add the following highlighted content:

```
resources:
```

```
  production_resource:
    type: OS::Heat::ResourceGroup
    properties:
      count: { get_param: instance_count }
      resource_def:
        type: Production::Custom::WebServer
        properties:
          instance_name:
            list_join: [ '', [ {get_param: instance_name}, "%index%" ] ]
          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 }
```

Save and exit the file.

6. Source the **/home/student/operator1-production-rc** identity environment file to launch the stack as `operator1`. Launch the stack using the **production-stack.yaml** template file and the **production-environment.yaml** environment file. Name the stack `production-webstack`.

   6.1. Use the `operator1` user credentials to perform a dry-run stack deployment and inspect the resources that will be created. Name the stack `production-webstack`. Use the **production-stack.yaml** template and the **production-environment.yaml** environment file to create the stack. Rectify any errors before proceeding to the next step.

> **NOTE**
> Before you perform a dry run of deploying the stack, download the `http://materials.example.com/heat/production-stack.yaml-final` template file to the **/home/student/heat-templates** directory. Use the **diff** command to compare the **production-stack.yaml** template file with the known good template file, **production-stack.yaml-final**. Fix any differences you find, and then proceed to launch the stack.

```
[student@workstation ~]$ cd ~/heat-templates
[student@workstation heat-templates]$ source ~/operator1-production-rc
[student@workstation heat-templates(operator1-production)]$ openstack stack \
create --environment production-environment.yaml \
--template production-stack.yaml \
--dry-run production-webstack
...output omitted...
```

   6.2. If the dry run succeeds, run the **openstack stack create** command. Do not use the **--dry-run** option while launching the stack.

```
[student@workstation heat-templates(operator1-production)]$ openstack stack \
create --environment production-environment.yaml \
--template production-stack.yaml \
--wait production-webstack
... [production-webstack]: CREATE_IN_PROGRESS  Stack CREATE started
... [production-webstack.production_resource]: CREATE_IN_PROGRESS  state changed
```

```
... [production-webstack.production_resource]: CREATE_COMPLETE  state changed
... [production-webstack]: CREATE_COMPLETE  Stack CREATE completed successfully
+---------------------+---------------------------------------+
| Field               | Value                                 |
+---------------------+---------------------------------------+
| id                  | 6964c8c2-0ac0-4f16-902a-8013773bc2f4  |
| stack_name          | production-webstack                   |
| description         | Launch multiple custom webservers     |
| creation_time       | 2018-09-25T13:37:10Z                  |
| updated_time        | None                                  |
| stack_status        | CREATE_COMPLETE                       |
| stack_status_reason | Stack CREATE completed successfully   |
+---------------------+---------------------------------------+
```

7. Verify that the `production-webstack` stack provisioned two instances. Use the **`instance_ipaddress`** output parameter of the `production-webstack` stack to retrieve the private IP addresses attached to the instances.

   7.1. View the instances launched by the orchestration stack. The IP addresses for your instances might vary.

```
[student@workstation heat-templates(operator1-production)]$ openstack server \
list -c Name -c Status -c Networks
+--------------------+--------+----------------------------------+
| Name               | Status | Networks                         |
+--------------------+--------+----------------------------------+
| production-server0 | ACTIVE | production-network1=192.168.1.107 |
| production-server1 | ACTIVE | production-network1=192.168.1.108 |
+--------------------+--------+----------------------------------+
```

   7.2. Use the **`instance_ipaddress`** output parameter of the `production-webstack` stack to retrieve the private IP address attached to the instances that were launched. The example output displays private IP addresses; `192.168.1.107`, and `192.168.1.108`, but the actual IP address might vary.

```
[student@workstation heat-templates(operator1-production)]$ openstack stack \
output show production-webstack instance_ipaddress
+--------------+-------------------------------------+
| Field        | Value                               |
+--------------+-------------------------------------+
| description  | private IP address of the webservers |
| output_key   | instance_ipaddress                  |
| output_value | [u'192.168.1.107', u'192.168.1.108'] |
+--------------+-------------------------------------+
```

Evaluation

On `workstation`, run the **`lab orchestration-review grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~] lab orchestration-review grade
```

Cleanup

On `workstation`, run the **`lab orchestration-review cleanup`** command to clean up this exercise.

```
[student@workstation ~] lab orchestration-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The Orchestration Service (heat) provides developers and system administrators an easy and repeatable way to create and manage a collection of related OpenStack resources.
- The Orchestration API service forwards requests to the Orchestration engine service using remote procedure calls (RPCs) over AMQP.
- The Orchestration engine service interprets the orchestration template and launches the stack.
- Using multiple layers of stacks that build on top of one another is the best way to organize an orchestration stack.
- You should perform a dry run of the stack deployment to verify any changes to infrastructure after updating a stack.
- Intrinsic functions in the Heat orchestration template assign values to properties that are available during the creation of a stack.
- Using the **`OS::Heat::SoftwareDeployment`** resource type allows any number of software configuration changes to be applied to an instance throughout its life cycle.
- When user data is changed and the orchestration stack is updated using the **`openstack stack update`** command, the instance is deleted and recreated using the updated user-data script.

CHAPTER 9

TROUBLESHOOTING OPENSTACK OPERATIONS

GOAL

Holistically diagnose and troubleshoot OpenStack issues.

OBJECTIVES

- Discuss recommended diagnostic and troubleshooting tools and techniques.
- Discuss common problem scenarios for selected core components.

SECTIONS

- Diagnosing OpenStack Issues (and Guided Exercise)
- Troubleshooting Common Core Issues (and Guided Exercise)

LAB

- Troubleshooting OpenStack Operations

DIAGNOSING OPENSTACK ISSUES

OBJECTIVES

After completing this section, students should be able to:

- Use recommended diagnostics and troubleshooting tools and techniques.
- Diagnose and troubleshoot issues with OpenStack containerized services.

BASIC TROUBLESHOOTING METHODOLOGY AND TOOLS

OpenStack operators and administrators might troubleshoot problems in more than one way. You need to find a methodology that works well to resolve the problem in the context in which you operate. To troubleshoot problems with an OpenStack service, it is important to understand the workflow that a request follows as it moves through the different components of the service. The following methodologies work well to resolve problems with OpenStack services:

- Verifying services: Ensure that the required services are up and running.
- Verifying ports: Ensure that the services are listening on the correct ports and IP addresses.
- OpenStack client debug: Run the OpenStack client in debug mode to look for the errors.
- Viewing logs: View log files for traces or errors.

General Tools for Troubleshooting

OpenStack is deployed in a Linux environment, which means that administrators can use popular Linux tools when troubleshooting. You can use the **ps**, **pgrep**, and many other commands to help verify the state of a process.

The **pgrep** command with the **-l** option lists the process name and their PIDs. For example, you can use the **pgrep -l nova** command to list the nova processes currently running.

```
[root@controller0 ~]# pgrep -l nova
12223 nova-consoleaut
12614 nova-novncproxy
13133 nova-conductor
13696 nova-scheduler
28382 nova-api-metada
```

The **watch** command provides a convenient way to execute a command at a given time interval. Use the **tail** command to output the last few lines of a file. You can use the **-f** option with the **tail** command to follow log files as they are updated.

Debugging Using the OpenStack Client

The OpenStack command-line client supports the **--debug** option to view more detail of errors. This option sets the logging level to **DEBUG** instead of the default **INFO** level. This debug information includes details of the API request sent by the OpenStack CLI and the response body details sent back from the API endpoint. This information contains useful pointers that can help you to troubleshoot issues.

```
[user@demo ~(user-demo)]$ openstack project create demo --debug
```

```
...output omitted...
Forbidden: You are not authorized to perform the requested action:
 identity:create_project. (HTTP 403) (Request-ID: req-7b9a8c8b-65a5-4819-8d50-
a550115b8c88)

END return value: 1
```

When a user runs a command, the operations can be recorded to identify resource changes and provide useful troubleshooting information. Use the **`--log-file`** option with the command to store the information in a log file.

```
[user@demo ~(user-demo)]$ openstack project create demo --debug --log-file
 demo.log
...output omitted...
```

Tools for Network Troubleshooting

For networking related issues, you need to verify the physical host to host communications between all the nodes to ensure the OpenStack control plane services are working properly. Operators and administrators must also verify that the overlay networks are communicating as intended. You can use the **`ip`**, **`tcpdump`**, **`arp`**, **`netstat`**, and many other commands to verify network connectivity.

Use **`ip addr`** to verify the status of the network interfaces attached to nodes.

```
[root@controller0 ~]# ip addr | grep " eth[0,1,2].*state"
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
 group default qlen 1000
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP group default qlen 1000
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP group default qlen 1000
```

You can also run the **`ssh`** command in verbose mode, which prints debugging information about SSH connection progress. This is useful for debugging connections, authentication, and any configuration problems when connecting to an instance. Use the **`-v`** option to turn on debugging mode while running SSH. There are different levels of verbosity; use multiple **`-v`** options to increase the verbosity (maximum verbosity level is 3).

Within the OpenStack control plane, the compute service communicates with the OpenStack networking service to create network ports and manage IP addresses on those ports. Without this communication, the instance might not receive an IP address from the DHCP server. Use the **`tcpdump`** command to dump traffic on the instance's network port.

Use the **`lsof`** command to find the processes listening on a particular port. This helps to determine if any other processes are running on the network port. The following output shows the noVNC compute service listening on port 6080.

```
[root@controller0 ~]# docker exec -ti nova_vnc_proxy lsof -i :6080 -S
COMMAND   PID USER   FD   TYPE DEVICE SIZE/OFF NODE NAME
nova-novn   1 nova    4u  IPv4 213361      0t0  TCP
 controller0.overcloud.example.com:6080 (LISTEN)
```

TRACING PACKETS IN OVN

With the implementation of ML2 using OVN, the OpenFlow rules manage outgoing and incoming traffic, security group implementation, DHCP, and NAT. These changes are especially important during troubleshooting. Network namespaces are no longer used for the DHCP agents and routers. The OVN Gateway links the overlay network, managed by `ovn-northd`, to the physical network. These are two types: a Layer 2 to bridge an OVN logical switch into a VLAN, and a Layer 3 providing a connection between an OVN router and the physical network.

OVN defines logical flows that have priorities, a match, and actions, generated by `ovn-northd`. Logical flows also describe the detailed behavior of an entire network. OVN creates the entire network in logical flows which are distributed to the `ovn-controller` on each hypervisor. Each `ovn-controller` translates the logical flows into OpenFlow flows, detailing how to reach the other hypervisors. Use the **`ovn-trace`** command to stimulate packet forwarding within an OVN logical network. The **`ovn-trace`** command takes two arguments: **DATAPATH**, which specifies the origin of the sample packet; and **MICROFLOW**, which specifies the sample packet to be simulated. The output of the command can have multiple formats with varying levels of detail.

While debugging a network connectivity issue to see why a packet is getting dropped, use the **`ovn-trace`** command to get a detailed trace of what happens to the packet originating from a logical port when sent to a destination. The following output shows the simulation of a DHCP request passing through logical networks. The packet originates from the instance port, and is broadcast to the DHCP port.

```
[root@controller0 ~]$ ovn-trace demo-network1 --summary\
'inport == "d05dd107-9226-405f-a77e-ec6fd4f80049" \
&& eth.src == fa:16:3e:ec:41:f5 \
&& eth.dst == ff:ff:ff:ff:ff:ff \
&& ip4.src == 0.0.0.0 \
&& ip4.dst == 255.255.255.255 \
&& udp.src == 68 && udp.dst == 67' \
...output omitted...
reg0[3] = put_dhcp_opts(offerip = 192.168.1.7, classless_static_route =
 {169.254.169.254/32, 192.168.1.2, 0.0.0.0/0, 192.168.1.1}, dns_server =
 {172.25.250.254}, lease_time = 43200, mtu = 1442, netmask = 255.255.255.0, router
 = 192.168.1.1, server_id = 192.168.1.1);
        /* We assume that this packet is DHCPDISCOVER or DHCPREQUEST. */;
        next;
        eth.dst = eth.src;
        eth.src = fa:16:3e:df:12:0c;
        ip4.dst = 192.168.1.7;
        ip4.src = 192.168.1.1;
        udp.src = 67;
        udp.dst = 68;
        outport = inport;
        flags.loopback = 1;
        output;
        egress(dp="demo-network1", inport="d05dd1", outport="d05dd1") {
            reg0[0] = 1;
            next;
            ct_next;
            ct_next(ct_state=est|trk /* default (use --ct to customize) */) {
                ct_commit;
                next;
                next;
                output;
                /* output to "d05dd1", type "" */;
```

```
...output omitted...
```

Verifying OpenStack Deployment Limits

When launching an instance, you might see errors because of the thresholds set to manage capacity and prevent over-utilization of resources based on project. Projects may be preconfigured with absolute limits and may differ in every OpenStack deployment. As an administrator, use the **`openstack limits show --absolute`** command to list the limits for a particular project.

```
[user@demo ~(admin-admin)]$ openstack limits show --absolute --project demo
+--------------------------+-------+
| Name                     | Value |
+--------------------------+-------+
| maxServerMeta            |   128 |
| maxTotalInstances        |    10 |
| maxPersonality           |     5 |
| totalServerGroupsUsed    |     0 |
| maxImageMeta             |   128 |
| maxPersonalitySize       | 10240 |
| maxTotalRAMSize          | 51200 |
| maxServerGroups          |    10 |
| maxSecurityGroupRules    |    20 |
| maxTotalKeypairs         |   100 |
| totalCoresUsed           |     2 |
| totalRAMUsed             |  2048 |
| maxSecurityGroups        |    10 |
| totalFloatingIpsUsed     |     0 |
| totalInstancesUsed       |     1 |
| maxServerGroupMembers    |    10 |
| maxTotalFloatingIps      |    10 |
| totalSecurityGroupsUsed  |     1 |
| maxTotalCores            |    20 |
| totalSnapshotsUsed       |     0 |
| maxTotalBackups          |    10 |
| maxTotalVolumeGigabytes  |  1000 |
| maxTotalSnapshots        |    10 |
| maxTotalBackupGigabytes  |  1000 |
| totalBackupGigabytesUsed |     0 |
| maxTotalVolumes          |    10 |
| totalVolumesUsed         |     0 |
| totalBackupsUsed         |     0 |
| totalGigabytesUsed       |     0 |
+--------------------------+-------+
```

Rate limits control the frequency at which users can issue specific API requests. Administrators can use rate limits to configure limits on the type and number of API calls that can be made in a specific time interval.

Quotas also provide a way to set thresholds on OpenStack resources. Quotas can be enforced on both project and the user accounts. You can manage quotas for the compute service, the block storage service, and the OpenStack networking service. Use the **`openstack quota show`** command to show the current quota values for a project.

```
[user@demo ~(admin-admin)]$ openstack quota show demo
+----------------------+-------+
```

```
| Field                 | Value |
+----------------------+-------+
| backup-gigabytes      | 1000  |
| backups               | 10    |
| cores                 | 20    |
| fixed-ips             | -1    |
| floating-ips          | 10    |
| gigabytes             | 1000  |
| groups                | 10    |
| injected-file-size    | 10240 |
| injected-files        | 5     |
| injected-path-size    | 255   |
| instances             | 10    |
| key-pairs             | 100   |
| per-volume-gigabytes | -1    |
| project               | demo  |
| project_name          | demo  |
| properties            | 128   |
| ram                   | 51200 |
| secgroup-rules        | 20    |
| secgroups             | 10    |
| snapshots             | 10    |
| volumes               | 10    |
+----------------------+-------+
```

VERIFYING OPENSTACK SERVICE STATUS

In the current version of Red Hat OpenStack Platform, the majority of the OpenStack services are no longer managed by the `systemd` utility. Most of these services run in their own execution environment inside a container, and these containerized services are commonly known as service containers. However, it is important to note that each service container does not map one-to-one to a particular container. Instead, these service containers are comprised of multiple discrete containerized processes. In a Red Hat OpenStack Platform deployment there are two types of service containers:

Non-HA containers
: These containers are managed and monitored using the `docker` daemon. All the non-HA containers are created and updated by **Paunch**. For example, containers for compute, identity, object storage, and many other non-HA OpenStack services.

HA containers
: These containers are created and monitored using `Pacemaker`. The HA containers are updated using Ansible Playbooks during the overcloud deployment. For example, `galera-bundle-docker-0`, `rabbitmq-bundle-docker-0`, and `redis-bundle-docker-0`, and other highly available OpenStack services.

Monitoring Containerized Services

To troubleshoot a failed service container on an overcloud node, start by checking the running containers on the host. These **`docker`** commands must be executed from the overcloud nodes. Use the **`docker ps`** command to determine the status of a service.

```
[root@controller0 ~]# docker ps --format="table {{.Names}}\t{{.Status}}"
...output omitted...
nova_scheduler                      Up 5 hours (healthy)
```

Use the **docker** subcommands such as **ps**, **history**, and **logs** to analyze and inspect the service containers. For example, use the **docker logs keystone** command to view the log for the `kesytone` container to determine what actually failed during the container run.

```
[root@controller0 ~]# docker logs keystone
+ sudo -E kolla_set_configs
INFO:__main__:Loading config file at /var/lib/kolla/config_files/config.json
INFO:__main__:Validating config file
INFO:__main__:Kolla config strategy set to: COPY_ALWAYS
INFO:__main__:Copying service configuration files
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/httpd/conf.d/10-
keystone_wsgi_admin.conf to /etc/httpd/conf.d/10-keystone_wsgi_admin.conf
INFO:__main__:Copying /var/lib/kolla/config_files/src/etc/httpd/conf.d/10-
keystone_wsgi_main.conf to /etc/httpd/conf.d/10-keystone_wsgi_main.conf
INFO:__main__:Deleting /etc/httpd/conf.d/ssl.conf
...output omitted...
```

To troubleshoot further you can inspect the container structure with the **docker inspect** command. You can use this command to inspect the following container characteristics:

- Different states of the container, such as **status**, **exitcodes**, **restartcount**.
- The container label, for more information, such as **version**, **build date**, and **config_id**.
- The storage, mounts, and networks used by the container.
- The health check commands used by the container to report the health status.

Use the following command to list the health check command for the `keystone` container.

```
[root@controller0 ~]# docker inspect keystone \
--format "{{json .Config.Healthcheck}}" | jq .
{
  "Test": [
    "CMD-SHELL",
    "/openstack/healthcheck"
  ]
}
```

In some cases, to inspect the container you might need to execute commands within a running container for further troubleshooting. For example, each container has a health check script to verify the service connection. Use the **docker exec** command to run the health check script from within the container.

```
[root@controller0 ~]# docker exec -ti keystone /openstack/healthcheck
{"versions": {"values": [{"status": "stable", "updated": "2018-02-28T00:00:00Z",
 "media-types": [{"base": "application/json", "type": "application/
vnd.openstack.identity-v3+json"}], "id": "v3.10", "links": [{"href":
 "http://controller0.internalapi.overcloud.example.com:5000/v3/", "rel":
 "self"}]}, {"status": "deprecated", "updated": "2016-08-04T00:00:00Z",
 "media-types": [{"base": "application/json", "type": "application/
vnd.openstack.identity-v2.0+json"}], "id": "v2.0", "links": [{"href":
 "http://controller0.internalapi.overcloud.example.com:5000/v2.0/", "rel":
 "self"}, {"href": "https://docs.openstack.org/", "type": "text/html", "rel":
 "describedby"}]}]}}
300 172.24.1.1:5000 0.006 seconds
```

When a container fails, many files on the local file system of the container are lost, because not all directories are bind mounted on the overcloud's host file system. Use the **`docker export`** command to export the container as a tar archive for future analysis.

```
[root@controller0 ~]# docker export keystone -o keystone.tar
```

Modifying Containerized Service Configuration Files

Container images used by a Red Hat OpenStack Platform deployment are derived from `Kolla` images. `Kolla` is an OpenStack project that provides production-ready containers and deployment tools for operating OpenStack cloud. Each container consists of the base image, which is further customized to add health check scripts, RHEL-specific packages, and TripleO packages to create configuration files using Ansible and Puppet. All the `Kolla` images contain a **`kolla_start`** bootstrap script that copies the configuration files from the host operating system to the container. The **`kolla_start`** script is executed during the initialization of a service within the container. The containerized service configuration managed by `Kolla` can take two forms, a JSON file or a directory tree.

The JSON file contains information such as the command to be run within the container, the directory containing the service configuration and logs, and the ownership that should be set on the log directory. The JSON files used are located on the host in the **`/var/lib/kolla/config_file`** directory. This file is bind mounted in the container as **`/var/lib/kolla/config_files/config.json`**.

In the output below, the `nova_scheduler` container bind mounts the **`/var/lib/kolla/config_files/nova_scheduler.json`** file in the container. The **`config.json`** file sets the permissions of the **`/var/log/nova`** directory, executes the **`nova_scheduler`** command, and copies the contents of **`/var/lib/kolla/config_files/src/*`** to the root directory within the container.

```
[root@controller0 ~]# docker inspect nova_scheduler \
--format "{{json .HostConfig.Binds}}" | jq 'map(select(. |
 contains("config.json")))'
[
  "/var/lib/kolla/config_files/nova_scheduler.json:/var/lib/kolla/config_files/
config.json:ro"
]
[root@controller0 ~]# cat /var/lib/kolla/config_files/nova_scheduler.json | jq .
{
  "permissions": [
    {
      "recurse": true,
      "path": "/var/log/nova",
      "owner": "nova:nova"
    }
  ],
  "command": "/usr/bin/nova-scheduler ",
  "config_files": [
    {
      "merge": true,
      "preserve_properties": true,
      "source": "/var/lib/kolla/config_files/src/*",
      "dest": "/"
    }
  ]
```

```
}
```

The directory tree contains a file hierarchy related to the service configuration and logs. It contains the **/etc** and **/var** directories. The directory specific to a service is located on the host in the **/var/lib/config-data/puppet-generated/*SERVICENAME*/** directory. The directory is bind mounted in the container as **/var/lib/kolla/config_files/src**. Bind mounted storage from the host provides persistent storage for the container's configuration, and log files. The **kolla_start** script copies contents from **/var/lib/kolla/config_files/src/** to the root directory (**/**) within the container.

```
[root@controller0 ~]# docker inspect nova_scheduler \
--format "{{json .HostConfig.Binds}}" | jq 'map(select(. |
 contains("config.json")))'
[
  "/var/lib/kolla/config_files/nova_scheduler.json:/var/lib/kolla/config_files/
config.json:ro",
  "/var/lib/config-data/puppet-generated/nova/:/var/lib/kolla/config_files/src:ro"
]
```

To modify the configuration in the containerized services that are managed by `Kolla`, you need to update the configuration file located in the **/var/lib/config-data/puppet-generated/*SERVICENAME*/** directory. The changes are not reflected immediately; you must restart the container to rerun the **kolla_start** script to apply the changes. Use the **docker restart** command to restart a container.

Viewing and Analyzing OpenStack Service Logs

Similar to configuration files, the logs from the containers are also bind mounted on the host operating system. Most containers store their logs under a specific directory in **/var/log/containers**. Containers using the `httpd` service store logs in the **httpd** subdirectory in **/var/log/containers**.

The output shows the log file for the `nova_placement` service.

```
[root@controller0 ~]# docker inspect nova_placement \
--format "{{json .HostConfig.Binds}}" \
| jq 'map(select(. | contains("config.json")))'
[
  "/var/log/containers/nova:/var/log/nova",
  "/var/log/containers/httpd/nova-placement:/var/log/httpd"
]
```

Use the **docker logs** command to inspect log files related to specific containers. The command output is the standard out and standard error for the container, not the service-related logs.

OpenStack services use the standard logging levels, at increasing severity: **DEBUG**, **INFO**, **AUDIT**, **WARNING**, **ERROR**, **CRITICAL**, and **TRACE**. These log messages only appear in the logs if they are more severe than the particular log level, with **DEBUG** allowing all log statements written in the log files. For example, **TRACE** is logged only if a specific software component involved has a stack trace, while **INFO** is logged for every message including those that are only for information.

The first step in finding the source of an error is typically to search for a **CRITICAL**, **TRACE**, or **ERROR** message in the log, starting at the bottom of the log file.

For example, when an instance creation fails, you will have to trace activities associated with the instance across the log files of various compute services, across both the controller and compute

nodes. The typical way is to trace the instance ID associated with an instance across the service logs. When dealing with OpenStack troubleshooting problems you need to keep in mind that because of the same underlying issue various OpenStack components write log entries across multiple log files.

Highly Available Containerized Services

In a conventional high-availability deployment, there are multiple controller nodes. Each controller node runs `Pacemaker` and `Corosync` to provide high availability services. Each node includes components managed by Pacemaker such as `RabbitMQ`, `HAProxy`, `Galera`, and `Redis`. These components are containerized core services supporting the major components of OpenStack. In each of these containers a `pacemaker_remote` daemon runs which provides local resource management for cluster resources and most command-line tools. This daemon allows a host to be used as a `Pacemaker` node without running the full cluster stack.

Each container running a specific service and belonging to different controller nodes in the cluster along with the `Pacemaker` resource constitutes a service bundle. You must *not* use the **`docker`** or the **`systemctl`** commands to manage these service bundles. The services managed by `Pacemaker` must always use the **`pcs`** commands to start or stop these service bundles.

Use the following command to list the containerized cluster services on the controller node:

```
[root@controller0 ~]# docker ps \
--format "table {{.Names}}\t{{.Image}}"| grep pcmklatest
openstack-manila-share-docker-0    172.25.249.200:8787/rhosp13/openstack-manila-
share:pcmklatest
openstack-cinder-volume-docker-0   172.25.249.200:8787/rhosp13/openstack-cinder-
volume:pcmklatest
haproxy-bundle-docker-0            172.25.249.200:8787/rhosp13/openstack-
haproxy:pcmklatest
redis-bundle-docker-0              172.25.249.200:8787/rhosp13/openstack-
redis:pcmklatest
galera-bundle-docker-0             172.25.249.200:8787/rhosp13/openstack-
mariadb:pcmklatest
rabbitmq-bundle-docker-0           172.25.249.200:8787/rhosp13/openstack-
rabbitmq:pcmklatest
```

Use the **`pcs status`** command to determine the service bundle status. The cluster resources that fail are reported under the **`Failed Actions`** heading of the output.

```
[root@controller0 ~]# pcs status
Cluster name: tripleo_cluster
Stack: corosync
Current DC: controller0 (version 1.1.18-11.el7_5.3-2b07d5c5a9) - partition with
 quorum
Last updated: Fri Oct 12 15:09:35 2018
Last change: Wed Oct 10 06:52:03 2018 by hacluster via crmd on controller0

5 nodes configured
21 resources configured

Online: [ controller0 ]
GuestOnline: [ galera-bundle-0@controller0 ovn-dbs-bundle-0@controller0 rabbitmq-
bundle-0@controller0 redis-bundle-0@controller0 ]

Full list of resources:
```

```
 Docker container: rabbitmq-bundle [172.25.249.200:8787/rhosp13/openstack-
rabbitmq:pcmklatest]
   rabbitmq-bundle-0  (ocf::heartbeat:rabbitmq-cluster):  Started controller0
 Docker container: galera-bundle [172.25.249.200:8787/rhosp13/openstack-
mariadb:pcmklatest]
   galera-bundle-0  (ocf::heartbeat:galera):  Master controller0
 Docker container: redis-bundle [172.25.249.200:8787/rhosp13/openstack-
redis:pcmklatest]
   redis-bundle-0 (ocf::heartbeat:redis): Master controller0
 ip-172.25.249.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.25.250.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.1.51 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.1.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.3.50 (ocf::heartbeat:IPaddr2): Started controller0
 ip-172.24.4.50 (ocf::heartbeat:IPaddr2): Started controller0
 Docker container: haproxy-bundle [172.25.249.200:8787/rhosp13/openstack-
haproxy:pcmklatest]
   haproxy-bundle-docker-0  (ocf::heartbeat:docker):  Started controller0
 Docker container: ovn-dbs-bundle [172.25.249.200:8787/rhosp13/openstack-ovn-
northd:latest]
   ovn-dbs-bundle-0 (ocf::ovn:ovndb-servers): Master controller0
 Docker container: openstack-cinder-volume [172.25.249.200:8787/rhosp13/openstack-
cinder-volume:pcmklatest]
   openstack-cinder-volume-docker-0 (ocf::heartbeat:docker):  Started controller0
 Docker container: openstack-manila-share [172.25.249.200:8787/rhosp13/openstack-
manila-share:pcmklatest]
   openstack-manila-share-docker-0  (ocf::heartbeat:docker):  Started controller0

Failed Actions:
* rabbitmq-bundle-docker-0_monitor_0 on controller0 'unknown error' (1): call=5,
 status=complete, exitreason='',
    last-rc-change='Fri Oct 12 07:29:15 2018', queued=0ms, exec=520ms
* galera-bundle-docker-0_monitor_0 on controller0 'unknown error' (1): call=9,
 status=complete, exitreason='',
    last-rc-change='Fri Oct 12 07:29:15 2018', queued=0ms, exec=493ms
Daemon Status:
  corosync: active/enabled
  pacemaker: active/enabled
  pcsd: active/enabled
```

The containers that are controlled by `Pacemaker` have the `RestartPolicy` variable set to **no** to ensure that Pacemaker controls the container start or stop actions. Use the following command to view the restart policy for the `haproxy-bundle-docker-0` container:

```
[root@controller0 ~]# docker inspect haproxy-bundle-docker-0 \
--format "{{json .HostConfig.RestartPolicy.Name}}"
"no"
```

Run the **`pcs resource show`** command to view more information of a resource bundle.

```
[root@controller0 ~]# pcs resource show haproxy-bundle
Bundle: haproxy-bundle
  Docker: image=172.25.249.200:8787/rhosp13/openstack-haproxy:pcmklatest
 network=host options="--user=root --log-driver=journald -e
```

```
 KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=1 run-command="/bin/bash /usr/local/
bin/kolla_start"
  Storage Mapping:
   options=ro source-dir=/var/lib/kolla/config_files/haproxy.json target-dir=/var/
lib/kolla/config_files/config.json (haproxy-cfg-files)
   options=ro source-dir=/var/lib/config-data/puppet-generated/haproxy/ target-
dir=/var/lib/kolla/config_files/src (haproxy-cfg-data)
   options=ro source-dir=/etc/hosts target-dir=/etc/hosts (haproxy-hosts)
   options=ro source-dir=/etc/localtime target-dir=/etc/localtime (haproxy-
localtime)
   options=rw source-dir=/var/lib/haproxy target-dir=/var/lib/haproxy (haproxy-
var-lib)
   options=ro source-dir=/etc/pki/ca-trust/extracted target-dir=/etc/pki/ca-trust/
extracted (haproxy-pki-extracted)
   options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-dir=/etc/pki/tls/
certs/ca-bundle.crt (haproxy-pki-ca-bundle-crt)
   options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-dir=/etc/
pki/tls/certs/ca-bundle.trust.crt (haproxy-pki-ca-bundle-trust-crt)
   options=ro source-dir=/etc/pki/tls/cert.pem target-dir=/etc/pki/tls/cert.pem
 (haproxy-pki-cert)
   options=rw source-dir=/dev/log target-dir=/dev/log (haproxy-dev-log)
```

TROUBLESHOOTING TOOLS FOR MESSAGING BROKER

One of the main components of OpenStack services is the messaging broker, which allows various OpenStack components to communicate internally using AMQP. Any message broker solution supporting AMQP can be used as a back end. Red Hat includes RabbitMQ as the message broker in its OpenStack product, because this provides enterprise-level features useful for setting up advanced configurations. Message brokers allow the sending and receiving of messages between producer and consumer applications. Internally, this communication is executed by RabbitMQ using exchanges, queues, and the bindings between those two. For example, for compute services, the `nova-api`, `nova-scheduler`, and `nova-compute` services use Remote Procedure Calls (RPCs) to communicate with one another using the message broker.

As a result, it is often useful to search for troubleshooting clues when diagnosing an issue with an OpenStack component. Refer to the section called "Troubleshooting Service Communication over Messaging" for more information on troubleshooting service communication using RabbitMQ.

REFERENCES

Further information about containerizing OpenStack services is available in the *Tips and Tricks for containerizing services* at
https://docs.openstack.org/tripleo-docs/latest/install/containers_deployment/

Further information on Paunch is available at
https://docs.openstack.org/paunch/latest/readme.html

Further information is available in the *Transitioning to Containerized Services* for Red Hat OpenStack Platform 13 at
https://access.redhat.com/documentation/en-us/red_hat_openstack_platform/13/

DIAGNOSING OPENSTACK ISSUES

In this exercise, you will walk through a proper diagnostic technique, look for log entries, query control plane services, and review the status of OpenStack components and services.

OUTCOMES

You should be able to locate and view log files, inspect services, and enable debug-level logging.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab troubleshooting-diagnose setup`** to ensure that the resources required to launch an instance are present.

```
[student@workstation ~]$ lab troubleshooting-diagnose setup
```

1. Examine the containerized Compute services running on `controller0`.

   1.1. Log in to `controller0` and change to the `root` user.

```
[student@workstation ~]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

   1.2. List the containerized Compute services.

```
[root@controller0 ~]# docker ps --filter "name=nova*" --all
CONTAINER ID        IMAGE
    COMMAND                  CREATED             STATUS                    PORTS
            NAMES
340500c7cd3b        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/usr/bin/bootstra..."   12 days ago         Exited (0) 12 days ago
            nova_api_discover_hosts
5bfcf5bbf99b        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "kolla_start"            12 days ago         Up 3 hours
            nova_metadata
457d4ace92a0        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "kolla_start"            12 days ago         Up 3 hours (healthy)
            nova_api
3a98c43a04f7        172.25.249.200:8787/rhosp13/openstack-nova-scheduler:latest
    "kolla_start"            12 days ago         Up 3 hours (healthy)
            nova_scheduler
35f522ad5783        172.25.249.200:8787/rhosp13/openstack-nova-novncproxy:latest
    "kolla_start"            12 days ago         Up 3 hours (healthy)
            nova_vnc_proxy
```

```
eb5d4133cd70        172.25.249.200:8787/rhosp13/openstack-nova-consoleauth:latest
    "kolla_start"            12 days ago         Up 3 hours (healthy)
            nova_consoleauth
a39f81aa4a8b        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "kolla_start"            12 days ago         Up 3 hours
            nova_api_cron
8ffb511c9318        172.25.249.200:8787/rhosp13/openstack-nova-conductor:latest
    "kolla_start"            12 days ago         Up 3 hours (healthy)
            nova_conductor
2178875e7453        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/usr/bin/bootstra..."   12 days ago         Exited (0) 12 days ago
            nova_db_sync
ee0b27e7e257        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/usr/bin/bootstra..."   12 days ago         Exited (0) 12 days ago
            nova_api_ensure_default_cell
b4eae39f8f50        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/usr/bin/bootstra..."   12 days ago         Exited (0) 12 days ago
            nova_api_map_cell0
53d487974cc1        172.25.249.200:8787/rhosp13/openstack-nova-placement-
api:latest   "kolla_start"            12 days ago         Up 3 hours
                     nova_placement
e77b4a296f8f        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/usr/bin/bootstra..."   12 days ago         Exited (0) 12 days ago
            nova_api_db_sync
d42de26be1af        172.25.249.200:8787/rhosp13/openstack-nova-placement-
api:latest   "/bin/bash -c 'cho..."   12 days ago         Exited (0) 12 days ago
                     nova_placement_init_log
166d1e71b4f3        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/bin/bash -c 'cho..."   12 days ago         Exited (0) 12 days ago
            nova_metadata_init_log
26874e77508d        172.25.249.200:8787/rhosp13/openstack-nova-api:latest
    "/bin/bash -c 'cho..."   12 days ago         Exited (0) 12 days ago
            nova_api_init_logs
```

2. On `workstation`, open a new terminal. Examine the containerized Compute services running on `compute0`.

2.1. Log in to `compute0` and change to the `root` user.

```
[student@workstation ~]$ ssh compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]#
```

2.2. List the containerized Compute services.

```
[root@compute0 ~]# docker ps --filter "name=nova*" --all
CONTAINER ID        IMAGE
 COMMAND                  CREATED             STATUS                  PORTS
         NAMES
6b487f0500b4        172.25.249.200:8787/rhosp13/openstack-nova-compute:latest
 "kolla_start"            12 days ago         Up 3 hours (healthy)
         nova_compute
9e0eeaf96ccd        172.25.249.200:8787/rhosp13/openstack-nova-compute:latest
 "kolla_start"            12 days ago         Up 3 hours
         nova_migration_target
```

```
3ff3bf523624        172.25.249.200:8787/rhosp13/openstack-nova-libvirt:latest   "/
bin/bash -c '/us..."   12 days ago         Exited (0) 12 days ago
      nova_libvirt_init_secret
635e1c629602        172.25.249.200:8787/rhosp13/openstack-nova-libvirt:latest
 "kolla_start"            12 days ago         Up 3 hours
         nova_libvirt
1eea921b6ea1        172.25.249.200:8787/rhosp13/openstack-nova-libvirt:latest
 "kolla_start"            12 days ago         Up 3 hours
         nova_virtlogd
```

3. On `compute0`, use the **`docker inspect`** command to determine the configuration and log file locations for the `nova_compute` service. List the configuration and log files.

   3.1. Determine the configuration and log file locations.

```
[root@compute0 ~]# docker inspect nova_compute
...output omitted...
        "HostConfig": {
            "Binds": [
                "/etc/pki/tls/certs/ca-bundle.crt:/etc/pki/tls/certs/ca-
bundle.crt:ro",
...output omitted...
                "/var/log/containers/nova:/var/log/nova",
                "/var/lib/kolla/config_files/nova_compute.json:/var/lib/kolla/
config_files/config.json:ro",
                "/var/lib/config-data/puppet-generated/nova_libvirt/:/var/lib/
kolla/config_files/src:ro",
                "/etc/iscsi:/var/lib/kolla/config_files/src-iscsid:ro"
            ],
...output omitted...
```

   3.2. List the configuration files.

```
[root@compute0 ~]# find /var/lib/config-data/puppet-generated/nova_libvirt
/var/lib/config-data/puppet-generated/nova_libvirt
/var/lib/config-data/puppet-generated/nova_libvirt/etc
/var/lib/config-data/puppet-generated/nova_libvirt/etc/libvirt
/var/lib/config-data/puppet-generated/nova_libvirt/etc/libvirt/libvirtd.conf
/var/lib/config-data/puppet-generated/nova_libvirt/etc/libvirt/passwd.db
/var/lib/config-data/puppet-generated/nova_libvirt/etc/libvirt/qemu.conf
/var/lib/config-data/puppet-generated/nova_libvirt/etc/my.cnf.d
/var/lib/config-data/puppet-generated/nova_libvirt/etc/my.cnf.d/tripleo.cnf
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/migration
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/migration/
authorized_keys
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/migration/identity
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/secret.xml
/var/lib/config-data/puppet-generated/nova_libvirt/etc/sasl2
/var/lib/config-data/puppet-generated/nova_libvirt/etc/sasl2/libvirt.conf
/var/lib/config-data/puppet-generated/nova_libvirt/etc/ssh
/var/lib/config-data/puppet-generated/nova_libvirt/etc/ssh/sshd_config
/var/lib/config-data/puppet-generated/nova_libvirt/var
/var/lib/config-data/puppet-generated/nova_libvirt/var/lib
/var/lib/config-data/puppet-generated/nova_libvirt/var/lib/nova
```

```
/var/lib/config-data/puppet-generated/nova_libvirt/var/lib/nova/.ssh
/var/lib/config-data/puppet-generated/nova_libvirt/var/lib/nova/.ssh/config
```

3.3. List the log files.

```
[root@compute0 ~]# find /var/log/containers/nova
/var/log/containers/nova
/var/log/containers/nova/privsep-helper.log
/var/log/containers/nova/nova-compute.log.4.gz
/var/log/containers/nova/nova-compute.log.3.gz
/var/log/containers/nova/nova-compute.log.2.gz
/var/log/containers/nova/nova-compute.log.1
/var/log/containers/nova/nova-compute.log
```

4. On `compute0`, enable **debug** level logging for the `nova_compute` service.

4.1. Change the logging configuration for `nova_compute`.

```
[root@compute0 ~]# crudini --set \
/var/lib/config-data/puppet-generated/nova_libvirt/etc/nova/nova.conf \
DEFAULT debug True
```

4.2. Restart the `nova_compute` service.

```
[root@compute0 ~]# docker restart nova_compute
nova_compute
```

4.3. View the **`/var/log/containers/nova/nova-compute.log`** file to verify that **debug** level logging is enabled.

```
[root@compute0 ~]# tail -f /var/log/containers/nova/nova-compute.log
...output omitted...
2018-11-05 05:55:24.829 1 DEBUG nova.compute.resource_tracker
 [req-1de591a0-8e6d-497d-b006-2743ab124a22 - - - - -] Compute_service record
 updated for compute0.overcloud.example.com:compute0.overcloud.example.com
 _update_available_resource /usr/lib/python2.7/site-packages/nova/compute/
resource_tracker.py:764
2018-11-05 05:55:24.830 1 DEBUG oslo_concurrency.lockutils
 [req-1de591a0-8e6d-497d-b006-2743ab124a22 - - - - -] Lock "compute_resources"
 released by "nova.compute.resource_tracker._update_available_resource" :: held
 1.309s inner /usr/lib/python2.7/site-packages/oslo_concurrency/lockutils.py:285
2018-11-05 05:55:24.830 1 DEBUG nova.service [req-1de591a0-8e6d-497d-
b006-2743ab124a22 - - - - -] Creating RPC server for service compute start /usr/
lib/python2.7/site-packages/nova/service.py:184
2018-11-05 05:55:24.865 1 DEBUG nova.service [req-1de591a0-8e6d-497d-
b006-2743ab124a22 - - - - -] Join ServiceGroup membership for this service compute
 start /usr/lib/python2.7/site-packages/nova/service.py:202
```

5. On `controller0`, force a failure in RabbitMQ by killing the appropriate `pacemaker_remoted` process.

5.1. Determine the ID of the RabbitMQ container.

```
[root@controller0 ~]# docker ps --filter "name=rabbit*"
```

```
CONTAINER ID        IMAGE
 COMMAND                  CREATED             STATUS              PORTS
    NAMES
0ffafb46725d        172.25.249.200:8787/rhosp13/openstack-rabbitmq:pcmklatest   "/
bin/bash /usr/lo..."   44 minutes ago      Up 44 minutes
 rabbitmq-bundle-docker-0
```

5.2. Obtain the PID for the `pacemaker_remoted` process.

```
[root@controller0 ~]# docker exec -it rabbitmq-bundle-docker-0 \
ps -efwww | grep pacemaker
root          20       1  0 Nov06 ?        00:00:03 /usr/sbin/pacemaker_remoted
```

5.3. Kill the `pacemaker_remoted` process in the container.

```
[root@controller0 ~]# docker exec -it rabbitmq-bundle-docker-0 \
kill -9 20
```

6. On `compute0`, review the **`/var/log/containers/nova/nova-compute.log`** file for activity related to the crash of the RabbitMQ container.

```
2018-11-07 00:31:31.914 1 ERROR oslo.messaging._drivers.impl_rabbit
 [req-4f29dc69-6f17-4695-a2b9-b8bf491ea69b - - - - -]
 [66da1b1d-092a-4761-8294-3e188d08ee43] AMQP server on
 controller0.internalapi.overcloud.example.com:5672 is unreachable: [Errno 111]
 ECONNREFUSED. Trying again in 4 seconds. Client port: None: error: [Errno 111]
 ECONNREFUSED
2018-11-07 00:31:33.049 1 ERROR oslo.messaging._drivers.impl_rabbit
 [req-4f29dc69-6f17-4695-a2b9-b8bf491ea69b - - - - -]
 [b6e28794-7a17-404c-a1b6-6f9123126c04] AMQP server on
 controller0.internalapi.overcloud.example.com:5672 is unreachable: [Errno 111]
 ECONNREFUSED. Trying again in 6 seconds. Client port: None: error: [Errno 111]
 ECONNREFUSED
2018-11-07 00:31:33.064 1 ERROR oslo.messaging._drivers.impl_rabbit
 [req-4f29dc69-6f17-4695-a2b9-b8bf491ea69b - - - - -] [6eb337d7-d0c6-4eeb-aeb6-
dd905d13358b] AMQP server on controller0.internalapi.overcloud.example.com:5672
 is unreachable: [Errno 111] ECONNREFUSED. Trying again in 4 seconds. Client port:
 None: error: [Errno 111] ECONNREFUSED
```

7. On `controller0`, determine the status of RabbitMQ.

7.1. Determine the ID of the RabbitMQ container. A change from the previous value indicates that Pacemaker has restarted the failed resource. Note that the uptime also indicates a restart.

```
[root@controller0 ~]# docker ps --filter "name=rabbit*"
CONTAINER ID        IMAGE
 COMMAND                  CREATED             STATUS              PORTS
    NAMES
```

```
0fe6e34b1d49        172.25.249.200:8787/rhosp13/openstack-rabbitmq:pcmklatest   "/
bin/bash /usr/lo..."   28 minutes ago      Up 28 minutes
 rabbitmq-bundle-docker-0
```

7.2. Determine the status of the Pacemaker cluster using the **`pcs status`** command. RabbitMQ is the messaging system connecting all components, so expect many failed actions.

```
[root@controller0 ~]# pcs status
Cluster name: tripleo_cluster
Stack: corosync
Current DC: controller0 (version 1.1.18-11.el7_5.3-2b07d5c5a9) - partition with
 quorum
Last updated: Wed Nov  7 01:02:22 2018
Last change: Mon Nov  5 00:44:01 2018 by root via cibadmin on controller0

5 nodes configured
21 resources configured

...output omitted...
Failed Actions:
* rabbitmq-bundle-docker-0_monitor_0 on controller0 'unknown error' (1): call=5,
 status=complete, exitreason='',
    last-rc-change='Tue Nov  6 23:21:47 2018', queued=0ms, exec=109ms
...output omitted...
```

8. Identify the Compute service processes listening on ports 8775, 8774, and 6080.

```
[root@controller0 ~]# ss -tnlp | egrep '8775|8774|6080'
LISTEN  0 128  172.25.250.50:6080  *:*  users:(("haproxy",pid=16308,fd=31))
LISTEN  0 128  172.24.1.50:6080    *:*  users:(("haproxy",pid=16308,fd=30))
LISTEN  0 100  172.24.1.1:6080     *:*  users:(("nova-novncproxy",pid=4998,fd=4))
LISTEN  0 128  172.25.250.50:8774  *:*  users:(("haproxy",pid=16308,fd=33))
LISTEN  0 128  172.24.1.50:8774    *:*  users:(("haproxy",pid=16308,fd=32))
LISTEN  0 128  172.24.1.1:8774     *:*  users:(("httpd",pid=20809,fd=3),
("httpd",pid=20808,fd=3),("httpd",pid=20771,fd=3),("httpd",pid=19531,fd=3),
("httpd",pid=17973,fd=3),("httpd",pid=10449,fd=3),("httpd",pid=10448,fd=3),
("httpd",pid=10447,fd=3),("httpd",pid=10446,fd=3),("httpd",pid=10445,fd=3),
("httpd",pid=10444,fd=3),("httpd",pid=10443,fd=3),("httpd",pid=10442,fd=3),
("httpd",pid=5539,fd=3))
LISTEN  0 128  172.24.1.1:8775     *:*  users:(("nova-api-metada",pid=22116,fd=4))
LISTEN  0 128  172.24.1.50:8775    *:*  users:(("haproxy",pid=16308,fd=29))
[root@controller0 ~]# logout
[heat-admin@controller0 ~]$ logout
[student@workstation ~]$
```

9. On `workstation`, list the Compute services.

9.1. Source the **`/home/student/architect1-finance-rc`** user environment file.

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

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

9.2. Execute the **`openstack compute service list`** command to view the status of the Compute services.

```
[student@workstation ~(architect1-finance)]$ openstack compute service list \
-f json
[
  {
    "Status": "enabled",
    "Binary": "nova-conductor",
    "Zone": "internal",
    "State": "up",
    "Host": "controller0.overcloud.example.com",
    "Updated At": "2018-11-07T02:50:30.000000",
    "ID": 1
  },
  {
    "Status": "enabled",
    "Binary": "nova-compute",
    "Zone": "nova",
    "State": "up",
    "Host": "compute1.overcloud.example.com",
    "Updated At": "2018-11-07T02:50:31.000000",
    "ID": 2
  },
  {
    "Status": "enabled",
    "Binary": "nova-compute",
    "Zone": "nova",
    "State": "up",
    "Host": "computehci0.overcloud.example.com",
    "Updated At": "2018-11-07T02:50:26.000000",
    "ID": 3
  },
  {
    "Status": "enabled",
    "Binary": "nova-compute",
    "Zone": "nova",
    "State": "up",
    "Host": "compute0.overcloud.example.com",
    "Updated At": "2018-11-07T02:50:21.000000",
    "ID": 4
  },
  {
    "Status": "enabled",
    "Binary": "nova-consoleauth",
    "Zone": "internal",
    "State": "up",
    "Host": "controller0.overcloud.example.com",
    "Updated At": "2018-11-07T02:50:28.000000",
    "ID": 5
  },
  {
    "Status": "enabled",
    "Binary": "nova-scheduler",
```

```
    "Zone": "internal",
    "State": "up",
    "Host": "controller0.overcloud.example.com",
    "Updated At": "2018-11-07T02:50:24.000000",
    "ID": 6
  }
]
```

10. On `workstation`, launch an instance to verify that all services are functional.

10.1. Source the **`/home/student/developer1-finance-rc`** user environment file.

```
[student@workstation ~(architect1-finance)]$ source ~/developer1-finance-rc
[student@workstation ~(developer1-finance)]$
```

10.2. Launch an instance named `finance-server1`, and use the **`--debug`** option to see detailed information.

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--image rhel7 \
--flavor default \
--security-group default \
--key-name example-keypair \
--nic net-id=finance-network1 \
--wait --debug finance-server1
...output omitted...
```

10.3. The debug output provides a wealth of information on all the processes and components involved in launching an instance. In the example below you can see that the call is to the `compute` service, the URL called, the resulting request ID, and the details of the request.

```
GET call to compute for http://172.25.250.50:8774/v2.1/servers/
f648a6ff-946e-409f-9316-e413013c4e42 used request id req-babca1b0-
ed9a-4e17-9356-4ce49f26bc9b

REQ: curl -g -i -X GET http://172.25.250.50:8774/v2.1/servers/
f648a6ff-946e-409f-9316-e413013c4e42 -H "User-Agent: python-
novaclient" -H "Accept: application/json" -H "X-Auth-Token:
 {SHA1}aa4da70ba16ede2fc5c4cc2e060c54d33f9d4464"
```

The debug output for calls made to other components will contain the JSON payloads where applicable.

Cleanup

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

```
[student@workstation ~]$ lab troubleshooting-diagnose cleanup
```

This concludes the guided exercise.

TROUBLESHOOTING COMMON CORE ISSUES

OBJECTIVES

After completing this section, students should be able to diagnose and troubleshoot the OpenStack networking, image, and volume services.

NETWORKING

This section discusses the different methods, commands, procedures and log files you can use to troubleshoot OpenStack networking issues.

Unreachable Instances

Problem: You have created an instance but are unable to assign it a floating IP address.

This problem can occur when the network is not configured correctly. If a router is not set as the gateway for the external network, then users will not be able to assign a floating IP address to an instance. Use the **`openstack router set --external-gateway`** ***`external_network router`*** command to set the router as a gateway for the external network. Use the **`openstack server add floating ip`** command to assign a floating IP address to the instance.

> **NOTE**
> You can create floating IP addresses even if the router is not connected to the external network but when the user attempts to associate a floating IP address with an instance, an error will be displayed.

```
[user@demo]$ openstack server add floating ip finance-web1 172.25.250.N
Error: External network 7aaf57c1-3c34-45df-94d3-dbc12754b22e is not reachable from
 subnet cfc7ddfa-4403-41a7-878f-e8679596eafd.
[user@demo]$ openstack router show finance-router1
+-------------------------------------------------------------+
| Field                   | Value                             |
+-------------------------------------------------------------+
| admin_state_up          | UP                                |
| availability_zone_hints |                                   |
| availability_zones      | nova                              |
| created_at              | 2017-06-15T09:39:07Z              |
| description             |                                   |
| external_gateway_info   | null                              |
| flavor_id               | None                              |
...output omitted...
```

Use the **`openstack router set --external-gateway`** ***`external_network router`*** command to set the router as a gateway for the external network.

```
[user@demo]$ openstack router set \
--external-gateway provider-datacentre finance-router1
[user@demo]$ openstack router show finance-router1 -f json
...output omitted...
```

```
  "external_gateway_info": "{\"network_id\": \"65606551-51f5-44f0-
a389-1c96b728e05f\", \"enable_snat\": true, \"external_fixed_ips\":
 [{\"subnet_id\": \"9d12d02f-7818-486b-8cbf-015798e28a4d\", \"ip_address\":
 \"172.25.250.32\"}]}",
...output omitted...
```

Use the **`openstack server add floating ip`** command to assign a floating IP address to the instance.

```
[user@demo]$ openstack server add floating ip finance-web1 172.25.250.N
```

Use the **`openstack server list`** command to verify that a floating IP address has been associated with the instance.

```
[user@demo]$ openstack server list -c Name -c Networks
+-----------------+--------------------------------------------+
| Name            | Networks                                   |
+-----------------+--------------------------------------------+
| finance-web1    | finance-network1=192.168.0.P, 172.25.250.N  |
+-----------------+--------------------------------------------+
```

Problem: You are unable to log in to the instance using SSH.

Check that an appropriate security group has been assigned to the instance and that a rule is present to allow SSH traffic. SSH connections are not allowed by default.

```
[user@demo]$ ssh cloud-user@172.25.250.N
ssh: connect to host 172.25.250.N port 22: Connection timed out
```

Verify that the security group assigned to the instance has a rule allowing SSH.

```
[user@demo]$ openstack server show finance-web1 -f json
...output omitted...
  "security_groups": "name='finance-web'",
...output omitted...
[user@demo]$ openstack security group rule list finance-web -f json
[
  {
    "IP Range": null,
    "Port Range": "",
    "Remote Security Group": null,
    "ID": "084a0216-0603-471f-ad1b-766e906a0a8f",
    "IP Protocol": null
  },
  {
    "IP Range": "0.0.0.0/0",
    "Port Range": "",
    "Remote Security Group": null,
    "ID": "38dd6907-893e-47e3-84a4-fe265d9629a2",
    "IP Protocol": "icmp"
  },
  {
    "IP Range": null,
    "Port Range": "",
```

```
    "Remote Security Group": null,
    "ID": "c0b6fc30-54c6-4f30-82d5-9fe8dcfe59b9",
    "IP Protocol": null
  }
]
[user@demo]$ openstack security group rule create --dst-port 22:22 finance-web
...output omitted...
[user@demo]$ openstack security group rule list finance-web -f json
...output omitted...
  {
    "IP Range": "0.0.0.0/0",
    "Port Range": "22:22",
    "Remote Security Group": null,
    "ID": "1f4c8ac6-8d36-4668-b578-1b8791c84d5e",
    "IP Protocol": "tcp"
  },
...output omitted...
```

This problem can also occur if **`cloud-init`** running on the instance was unable to communicate with the Metadata service, resulting in the SSH public key not being added to the **`authorized_keys`** file.

Communication between the instance and the Metadata service involves several components, however a common cause of this failure is that the project network does not have a router attached and the **`NeutronEnableIsolatedMetadata`** option was not enabled during overcloud deployment.

You can verify this by checking the **`/var/log/cloud-init.log`** log file on the instance itself. Alternatively, check the contents of the **`/home/cloud-user/.ssh/authorized-keys`** file. Access the instance directly through the console URL or through the dashboard.

```
[root@host-192-168-0-N ~]# less /var/log/cloud-init.log
...output omitted...
[ 134.170335] cloud-init[475]: 2014-07-01 07:33:22,857 -
url_helper.py[WARNING]: Calling 'http://192.168.0.1//latest/meta-data/instance-
id' failed [0/120s]:
request error [HTTPConnectionPool(host='192.168.0.1', port=80): Max retries
exceeded with url: //latest/meta-data/instance-id (...)
[Errno 113] No route to host)]
...output omitted...
[root@host-192-168-0-N ~]# cat /home/cloud-user/.ssh/authorized-keys
[root@host-192-168-0-N ~]#
```

In this situation, attach the subnet to a router and then reboot the instance.

```
[user@demo]$ openstack subnet list
+---------------+---------------------------+---------------+-----------------+
| ID            | Name                      | Network       | Subnet          |
+---------------+---------------------------+---------------+-----------------+
| 72c4(...)cc37 | provider-subnet-172.25.250 | 8b00(...)5285 | 172.25.250.0/24 |
| a520(...)1d9a | finance-subnet1           | f33a(...)42b2 | 192.168.0.0/24  |
+---------------+---------------------------+---------------+-----------------+
[user@demo]$ openstack router add subnet finance-router1 finance-subnet1
[user@demo]$ openstack port list --router finance-router1 \
```

```
-c 'Fixed IP Addresses'
+-------------------------------------------------------------+
| Fixed IP Addresses                                          |
+-------------------------------------------------------------+
| ip_address='192.168.0.1', subnet_id='f33a(...)42b2'         |
+-------------------------------------------------------------+
[user@demo]$ openstack server reboot finance-web1
```

Another cause of SSH failure is that a key pair was not assigned to the instance when created. In this scenario, the instance must be destroyed and recreated with a key pair assigned.

```
[user@demo]$ openstack server delete finance-web1
[user@demo]$ openstack server create \
--flavor default \
--nic net-id=finance-network1 \
--key-name example-keypair \
--security-group finance-web \
--image rhel7 finance-web1 --wait
[user@demo]$ openstack server show finance-web1 -f json
...output omitted...
  "key_name": "example-keypair",
...output omitted...
```

Physical Network Issues

Problem: You have verified the configuration of all OpenStack network components, however communication between compute nodes or to systems outside of OpenStack is still failing.

All OpenStack software-defined networks depend on the underlying physical networks. Physical network connections can be tested from the controller or compute nodes using several tools.

Firstly, determine which physical interface to investigate. Use the **`ovs-vsctl`** command to discover the mapping of physical interfaces to OVS bridges.

```
[root@controller0 ~]# ovs-vsctl show
6c237e8b-5014-42f1-9ac4-cc7728797597
    Bridge br-trunk
        fail_mode: standalone
        Port "vlan50"
            tag: 50
            Interface "vlan50"
                type: internal
        Port "vlan40"
            tag: 40
            Interface "vlan40"
                type: internal
        Port "vlan30"
            tag: 30
            Interface "vlan30"
                type: internal
        Port "vlan20"
            tag: 20
            Interface "vlan20"
                type: internal
        Port "eth1"
```

```
            Interface "eth1"
        Port br-trunk
            Interface br-trunk
                type: internal
        Port "vlan10"
            tag: 10
            Interface "vlan10"
                type: internal
    Bridge "br-eth4"
        fail_mode: standalone
        Port "br-eth4"
            Interface "br-eth4"
                type: internal
        Port "eth4"
            Interface "eth4"
    Bridge "br-eth3"
        fail_mode: standalone
        Port "eth3"
            Interface "eth3"
        Port "br-eth3"
            Interface "br-eth3"
                type: internal
...output omitted...
    Bridge br-ex
        fail_mode: standalone
        Port "patch-provnet-fc5472ee-98d9-4f6b-9bc9-544ca18aefb3-to-br-int"
            Interface "patch-provnet-fc5472ee-98d9-4f6b-9bc9-544ca18aefb3-to-br-
int"
                type: patch
                options: {peer="patch-br-int-to-provnet-
fc5472ee-98d9-4f6b-9bc9-544ca18aefb3"}
        Port "eth2"
            Interface "eth2"
        Port br-ex
            Interface br-ex
                type: internal
    ovs_version: "2.9.0"
```

`br-int` has no physical interfaces; they are all internal. From the output above, the bridge to interface mappings are:

BRIDGE	INTERFACES
br-trunk	eth1
br-ex	eth2
br-eth3	eth3
br-eth4	eth4

Use the **`ip link show`** command to verify the physical connections.

```
[root@controller0 ~]# ip link show | grep ' eth[1-4]'
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP mode DEFAULT group default qlen 1000
```

```
4: eth2: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP mode DEFAULT group default qlen 1000
5: eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP mode DEFAULT group default qlen 1000
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system state UP mode DEFAULT group default qlen 1000
```

The **LOWER_UP** setting means that a link is present. The MTU is also shown in the output. MTU mismatches can cause dropped or fragmented packets.

To analyze traffic entering and leaving an interface, use the **tcpdump** command. **tcpdump** can show whether packets are being fragmented, retransmitted, or only working in one direction. The following example highlights capturing packets with VLAN tags.

```
[root@controller0 ~]# tcpdump -nnei eth1 -vvv
06:12:38.660519 62:b4:61:50:de:7f > 12:24:ea:3c:0c:60, ethertype 802.1Q (0x8100),
 length 70: vlan 10, p 0, ethertype IPv4, (tos 0x0, ttl 64, id 14543, offset 0,
 flags [DF], proto TCP (6), length 52)
    172.24.1.1.5672 > 172.24.1.2.51802: Flags [.], cksum 0x5a5a (incorrect ->
 0x415c), seq 0, ack 1, win 947, options [nop,nop,TS val 12358080 ecr 11722736],
 length 0
```

OpenFlow Rule Analysis

Problem: you suspect that packets are not being processed correctly through OVN.

With OVN providing Layer 3 services, the OpenFlow rules are now more numerous than with OVS. Tracing packets through OVN manually by dumping the flows is a tedious and potentially error-prone operation. Packets are tested against the rules in each table, possibly modified, then resubmitted to the next table.

The **ovs-appctl ofproto/trace** command allows the simulation of different packet types entering an OVN bridge on a given interface. To trace packets from a specific instance, first obtain the MAC address and TAP interface name. From the applicable compute host, use the **virsh dumpxml *domain*** command.

```
[root@compute0 ~]# virsh dumpxml instance-00000008
...output omitted...
    <interface type='bridge'>
      <mac address='fa:16:3e:79:0c:05'/>
      <source bridge='br-int'/>
      <virtualport type='openvswitch'>
        <parameters interfaceid='10447eb8-a243-4c61-877f-1f9a5f3c553e'/>
      </virtualport>
      <target dev='tap10447eb8-a2'/>
      <model type='virtio'/>
      <mtu size='1442'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
...output omitted...
```

The first argument to the **ovs-appctl ofproto/trace** command is the bridge name, and the second is a collection of fields describing the simulated packet. The following table describes the fields and their values used in the example below.

Packet Fields

FIELD	DESCRIPTION	VALUE
in_port=	The port the packet is entering the bridge on.	tap10447eb8-a2
protocol	The protocol of the packet.	**tcp**
nw_src=	The source network address (the IP address of our instance).	192.168.1.11
dl_src=	The source Ethernet address (the MAC address of our instance)	fa:16:3e:79:0c:05
nw_dst=	The destination network address (the IP address of our target).	192.168.1.8
tcp_dst=	The destination TCP port.	22

NOTE
The command input and output below has been broken across lines for readability, do not include carriage returns or spaces in the packet descriptor when typing.

```
[root@compute0 ~]# ovs-appctl ofproto/trace br-int \
in_port="tap10447eb8-a2",tcp,nw_src=192.168.1.11,dl_src=fa:16:3e:79:0c:05,
nw_dst=192.168.1.8,tcp_dst=22
Flow:
 tcp,in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:79:0c:05,dl_dst=00:00:00:00:00:00,
nw_src=192.168.1.11,nw_dst=192.168.1.8,nw_tos=0,nw_ecn=0,nw_ttl=0,tp_src=0,
tp_dst=22,tcp_flags=0

bridge("br-int")
----------------
 0. in_port=4, priority 100
    set_field:0x1->reg13
    set_field:0x3->reg11
    set_field:0x2->reg12
    set_field:0x5->metadata
    set_field:0x6->reg14
    resubmit(,8)
 8. reg14=0x6,metadata=0x5,dl_src=fa:16:3e:79:0c:05, priority 50, cookie
 0xa7713621
    resubmit(,9)
 9. ip,reg14=0x6,metadata=0x5,dl_src=fa:16:3e:79:0c:05,nw_src=192.168.1.11,
 priority 90, cookie 0x88ad980
    resubmit(,10)
10. metadata=0x5, priority 0, cookie 0x2492510a
    resubmit(,11)
11. ip,metadata=0x5, priority 100, cookie 0x1548770e
    load:0x1->NXM_NX_XXREG0[96]
    resubmit(,12)
12. metadata=0x5, priority 0, cookie 0x662626bf
    resubmit(,13)
```

```
13. ip,reg0=0x1/0x1,metadata=0x5, priority 100, cookie 0x99fcdd84
    ct(table=14,zone=NXM_NX_REG13[0..15])
    drop
     -> A clone of the packet is forked to recirculate. The forked pipeline will
 be resumed at table 14.

Final flow:
 tcp,reg0=0x1,reg11=0x3,reg12=0x2,reg13=0x1,reg14=0x6,metadata=0x5,in_port=4,
vlan_tci=0x0000,dl_src=fa:16:3e:79:0c:05,dl_dst=00:00:00:00:00:00,
nw_src=192.168.1.11,nw_dst=192.168.1.8,nw_tos=0,nw_ecn=0,nw_ttl=0,
tp_src=0,tp_dst=22,tcp_flags=0
Megaflow: recirc_id=0,eth,ip,in_port=4,vlan_tci=0x0000/0x1000,
dl_src=fa:16:3e:79:0c:05,nw_src=192.168.1.11,nw_frag=no
Datapath actions: ct(zone=1),recirc(0x26)

===============================================================================
recirc(0x26) - resume conntrack with default ct_state=trk|new (use --ct-next to
 customize)
===============================================================================

Flow: recirc_id=0x26,ct_state=new|
trk,ct_zone=1,eth,tcp,reg0=0x1,reg11=0x3,reg12=0x2,
reg13=0x1,reg14=0x6,metadata=0x5,in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:79:0c:05,
dl_dst=00:00:00:00:00:00,nw_src=192.168.1.11,nw_dst=192.168.1.8,nw_tos=0,nw_ecn=0,
nw_ttl=0,tp_src=0,tp_dst=22,tcp_flags=0

bridge("br-int")
----------------
    thaw
        Resuming from table 14
14. ct_state=+new-est+trk,ip,reg14=0x6,metadata=0x5, priority 2002, cookie
 0xc37ad129
    load:0x1->NXM_NX_XXREG0[97]
    resubmit(,15)
15. metadata=0x5, priority 0, cookie 0xcd053aee
    resubmit(,16)
16. metadata=0x5, priority 0, cookie 0x73831053
    resubmit(,17)
17. metadata=0x5, priority 0, cookie 0x581d089e
    resubmit(,18)
18. ip,reg0=0x2/0x2,metadata=0x5, priority 100, cookie 0x9e9123b2
    ct(commit,zone=NXM_NX_REG13[0..15],exec(load:0->NXM_NX_CT_LABEL[0]))
    load:0->NXM_NX_CT_LABEL[0]
    resubmit(,19)
19. metadata=0x5, priority 0, cookie 0xe4207168
    resubmit(,20)
20. metadata=0x5, priority 0, cookie 0x5de29887
    resubmit(,21)
21. metadata=0x5, priority 0, cookie 0xa4953307
    resubmit(,22)
22. metadata=0x5, priority 0, cookie 0xf2598275
    resubmit(,23)
23. metadata=0x5, priority 0, cookie 0xed722019
    resubmit(,24)
24. No match.
    drop
```

```
Final flow:
 recirc_id=0x26,eth,tcp,reg0=0x3,reg11=0x3,reg12=0x2,reg13=0x1,reg14=0x6,
metadata=0x5,in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:79:0c:05,
dl_dst=00:00:00:00:00:00,nw_src=192.168.1.11,nw_dst=192.168.1.8,
nw_tos=0,nw_ecn=0,nw_ttl=0,tp_src=0,tp_dst=22,tcp_flags=0
Megaflow: recirc_id=0x26,ct_state=+new-est-rel-rpl-inv
+trk,ct_label=0/0x1,eth,ip,in_port=4,dl_src=fa:16:3e:79:0c:05,
dl_dst=00:00:00:00:00:00,nw_dst=192.168.1.8/29,nw_frag=no
Datapath actions: ct(commit,zone=1,label=0/0x1)
```

The end result is that the packet is dropped, which is correct because the target instance does not exist.

IMAGES

The OpenStack image service stores images and metadata. Users can create images and upload them to the image service. The image service has a RESTful API that allows users to query the metadata of an image, as well as obtaining the actual image.

Logging

Logging for the image service can be configured in the **`/var/lib/config-data/puppet-generated/glance_api/etc/glance/glance-api.conf`** configuration file. It allows the configuration of the logging level, the log file location, and whether to use `syslog` or `journald`.

The maximum size for an image can be restricted using the **`image_size_cap`** option, and storage capacity limits per user can be configured with the **`user_storage_quota`** option.

Managing Images

When creating a new image with the **`openstack`** command, you can choose to protect that image from deletion with the **`--protected`** option. This prevents an image from being deleted even by the administrator. The image must be unprotected before you can delete it.

```
[user@demo]$ openstack image delete rhel7-web
Failed to delete image with name or ID '21b3b8ba-e28e-4b41-9150-ac5b44f9d8ef':
403 Forbidden
Image 21b3b8ba-e28e-4b41-9150-ac5b44f9d8ef is protected and cannot be deleted.
(HTTP 403)
Failed to delete 1 of 1 images.
[user@demo]$ openstack image set --unprotected rhel7-web
[user@demo]$ openstack image delete rhel7-web
```

VOLUMES

Ceph

The OpenStack block storage service can use Ceph as a storage back end. Each volume created in the block storage service has an associated RBD image in Ceph. The name of the RBD image is the ID of the block storage volume.

The OpenStack block storage service requires a user and a pool in Ceph in order to use it. The user is `openstack`, the same user configured for other services using Ceph as their back end, like the OpenStack image service. The undercloud also creates a dedicated Ceph pool for the block storage services, named `volumes`. The `volumes` pool contains all the RBD images associated with volumes. These settings are included in the **`/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf`** configuration file.

```
[user@demo]$ grep ^rbd_ \
/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf
rbd_pool=volumes
rbd_user=openstack
...output omitted...
```

Permissions within Ceph are known as capabilities, and are granted by daemon type, such as MON or OSD. Three capabilities are available within Ceph: read (r) to view, write (w) to modify, and execute (x) to execute extended object classes. All daemon types support these three capabilities. For the OSD daemon type, permissions can be restricted to one or more pools, for example:

```
osd 'allow rwx pool=rbd, allow rx pool=mydata'
```

If no pool is specified, the permission is granted on all existing pools. The `openstack` user has capabilities on all the pools used by OpenStack services.

The `openstack` user requires read, write, and execute capabilities in both the `volumes` and the `images` pools to be used by the OpenStack block storage service. The `images` pool is dedicated to the OpenStack image service.

```
[user@demo]$ ceph auth list
installed auth entries:
...output omitted...
client.openstack
    key: AQCg7T5ZAAAAABAAI6ZtsCQEuvVNqoyRKzeNcw==
        caps: [mon] allow r
            caps: [osd] allow class-read object_prefix rbd_children,
            allow rwx pool=volumes, allow rwx pool=backups, allow rwx pool=vms,
            allow rwx pool=images, allow rwx pool=metrics
```

Attach and Detach Workflow

There are three API calls for each attach and detach operation.

- Status of the volume is updated in the database
- Connection operations on the volume are handled
- Status of the volume is finalized and the resource is released

In order to attach a volume, it must be in the **`available`** state. Any other state results in an error message. It can happen that a volume is stuck in a **`detaching`** state. The state can be altered by an admin user.

```
[user@demo]$ cinder reset-state --state available volume_id
```

If you try to delete a volume and it fails, you can forcefully delete that volume using the **`--force`** option.

```
[user@demo]$ openstack volume delete --force volume_id
```

Incorrect volume configurations are the most common cause of block storage errors. Consult the OpenStack block storage service log files in case of error.

Log Files

Block Storage Log File Locations

SERVICE	LOG PATH
API service	**`/var/log/containers/cinder/cinder-api.log`**
volume service	**`/var/log/containers/cinder/cinder-volume.log`**
scheduler service	**`/var/log/containers/cinder/cinder-scheduler.log`**

The block storage service API log file is useful in determining whether the error is due to an endpoint or connectivity error. That is, if you try to create a volume and it fails, then the **`/var/log/containers/cinder/cinder-api.log`** file should be reviewed first. Verify that the request was received in the **`/var/log/containers/cinder/cinder-api.log`** file. Assuming that there were no errors, check the **`/var/log/containers/cinder/cinder-volume.log`** for errors that may have occurred during volume creation.

For OpenStack block services to function properly, they must be configured to use the RabbitMQ messaging service. All configuration can be found in the **`/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf`** configuration file, stored on the controller node. The default value for the **`rabbit_userid`** option is **`guest`**. If that user is incorrectly configured and the block storage services are restarted, RabbitMQ will not respond to requests because authentication will fail. Any volume created during that period results in an **ERROR** status. Any volume with a status of **ERROR** must be deleted and recreated after the block storage service has been restarted and is running properly.

To determine the problem in this scenario, review the **`/var/log/containers/cinder/cinder-scheduler.log`** log file on the controller node. If the problem is RabbitMQ, you will see the following:

```
...output omitted...
201 (...) Failed to run task
cinder.scheduler.flows.create_volume.ScheduleCreateVolumeTask;volume:create:
No valid host was found. No weighed hosts available
```

Verify that both the RabbitMQ cluster and the `rabbitmq-bundle` Pacemaker resource are available. If both resources are available the problem might be found in the **`/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf`** configuration file. Ensure that all usernames, passwords, IP addresses, and URLs in the **`/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf`** configuration file are correct.

```
[user@demo]$ docker exec -t \
rabbitmq-bundle-docker-0 rabbitmqctl status
...output omitted...
{listeners,[{clustering,25672,"::"},{amqp,5672,"172.24.1.1"}]},
...output omitted...
[user@demo]$ sudo pcs resource show rabbitmq-bundle
 Bundle: rabbitmq-bundle
  Docker: image=172.25.249.200:8787/rhosp13/openstack-rabbitmq:pcmklatest
 network=host options="--user=root --log-driver=journald -e
```

```
 KOLLA_CONFIG_STRATEGY=COPY_ALWAYS" replicas=1 run-command="/bin/bash /usr/local/
bin/kolla_start"
  Network: control-port=3122
  Storage Mapping:
   options=ro source-dir=/var/lib/kolla/config_files/rabbitmq.json target-dir=/
var/lib/kolla/config_files/config.json (rabbitmq-cfg-files)
   options=ro source-dir=/var/lib/config-data/puppet-generated/rabbitmq/ target-
dir=/var/lib/kolla/config_files/src (rabbitmq-cfg-data)
   options=ro source-dir=/etc/hosts target-dir=/etc/hosts (rabbitmq-hosts)
   options=ro source-dir=/etc/localtime target-dir=/etc/localtime (rabbitmq-
localtime)
   options=rw source-dir=/var/lib/rabbitmq target-dir=/var/lib/rabbitmq (rabbitmq-
lib)
   options=ro source-dir=/etc/pki/ca-trust/extracted target-dir=/etc/pki/ca-trust/
extracted (rabbitmq-pki-extracted)
   options=ro source-dir=/etc/pki/tls/certs/ca-bundle.crt target-dir=/etc/pki/tls/
certs/ca-bundle.crt (rabbitmq-pki-ca-bundle-crt)
   options=ro source-dir=/etc/pki/tls/certs/ca-bundle.trust.crt target-dir=/etc/
pki/tls/certs/ca-bundle.trust.crt (rabbitmq-pki-ca-bundle-trust-crt)
   options=ro source-dir=/etc/pki/tls/cert.pem target-dir=/etc/pki/tls/cert.pem
 (rabbitmq-pki-cert)
   options=rw source-dir=/var/log/containers/rabbitmq target-dir=/var/log/rabbitmq
 (rabbitmq-log)
   options=rw source-dir=/dev/log target-dir=/dev/log (rabbitmq-dev-log)
  Resource: rabbitmq (class=ocf provider=heartbeat type=rabbitmq-cluster)
   Attributes: set_policy="ha-all ^(?!amq\.).* {"ha-mode":"all"}"
   Meta Attrs: container-attribute-target=host notify=true
   Operations: monitor interval=10 timeout=40 (rabbitmq-monitor-interval-10)
               start interval=0s timeout=200s (rabbitmq-start-interval-0s)
               stop interval=0s timeout=200s (rabbitmq-stop-interval-0s)
```

REFERENCES

Further information is available 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/

ovs-fields(7) man page

TROUBLESHOOTING COMMON CORE ISSUES

In this exercise, you will fix an issue related to launching instances. You will also fix an issue with network communication.

OUTCOMES

You should be able to:

- Troubleshoot and fix an issue causing instance launch to fail.
- Troubleshoot and fix an issue in the OpenStack Networking service.

BEFORE YOU BEGIN

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

On `workstation`, run **`lab troubleshooting-services setup`** to prepare the environment for this exercise, and to ensure all requires resources are present.

```
[student@workstation ~]$ lab troubleshooting-services setup
```

1. Launch an instance named `finance-server1`.

   1.1. Source the **`/home/student/developer1-finance-rc`** user environment file.

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

   1.2. Create an instance named `finance-server1` using the following values.

OPTION	VALUE
`--flavor`	default
`--image`	rhel7
`--key-name`	example-keypair
`--network`	provider1-104

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--network provider1-104 \
--wait \
finance-server1
```

The command hangs.

1.3. Cancel the deployment using **Ctrl+C**, delete the instance, and then relaunch using the **--debug** option.

```
[student@workstation ~(developer1-finance)]$ openstack server delete \
finance-server1
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--network provider1-104 \
--wait \
finance-server1 --debug
...output omitted...
GET call to compute for http://172.25.250.50:8774/v2.1/servers/1619be15-a922-4e3f-
afd1-afbdc734fb31 used request id req-d04865fe-9fa7-4e74-8a83-d6b1d98cf640
REQ: curl -g -i -X GET http://172.25.250.50:8774/v2.1/servers/1619be15-a922-4e3f-
afd1-afbdc734fb31 -H "User-Agent: python-novaclient" -H "Accept: application/json"
 -H "X-Auth-Token: {SHA1}41d0ee7cf0d8faa6e83c22930185063deb01960b"
http://172.25.250.50:8774 "GET /v2.1/servers/1619be15-a922-4e3f-afd1-afbdc734fb31
 HTTP/1.1" 200 609
RESP: [200] Date: Tue, 13 Nov 2018 02:38:54 GMT Server: Apache OpenStack-
API-Version: compute 2.1 X-OpenStack-Nova-API-Version: 2.1 Vary: OpenStack-
API-Version,X-OpenStack-Nova-API-Version,Accept-Encoding x-openstack-request-
id: req-afaf9cca-4811-4be5-85ba-7869e22cb9e0 x-compute-request-id: req-
afaf9cca-4811-4be5-85ba-7869e22cb9e0 Content-Encoding: gzip Content-Length: 609
 Content-Type: application/json
RESP BODY: {"server": {"OS-EXT-STS:task_state": "scheduling", "addresses": ...,
 "links": [{"href": "http://172.25.250.50:8774/v2.1/servers/1619be15-a922-4e3f-
afd1-afbdc734fb31", "rel": "self"}, {"href": "http://172.25.250.50:8774/
servers/1619be15-a922-4e3f-afd1-afbdc734fb31", "rel": "bookmark"}],
 "image": {"id": "6b0128a9-4481-4ceb-b34e-ffe92e0dcfdd", "links": [{"href":
 "http://172.25.250.50:8774/images/6b0128a9-4481-4ceb-b34e-ffe92e0dcfdd", "rel":
 "bookmark"}]}, "OS-EXT-STS:vm_state": "building", "OS-SRV-USG:launched_at":
 null, "flavor": {"id": "e04380ed-b027-4a72-a697-4307bc014b6c", "links": [{"href":
 "http://172.25.250.50:8774/flavors/e04380ed-b027-4a72-a697-4307bc014b6c",
 "rel": "bookmark"}]}, "id": "1619be15-a922-4e3f-afd1-afbdc734fb31", "user_id":
 "e4035d555f6b88cf42ca4cacb9fa9999dca9787392222d2eb0875e4e34e6d76f", "OS-
DCF:diskConfig": "MANUAL", "accessIPv4": "", "accessIPv6": "", "progress":
 0, "OS-EXT-STS:power_state": 0, "OS-EXT-AZ:availability_zone": "",
 "config_drive": "", "status": "BUILD", "updated": "2018-11-13T02:38:54Z",
 "hostId": "", "OS-SRV-USG:terminated_at": null, "key_name": "example-keypair",
 "name": "finance-server1", "created": "2018-11-13T02:37:37Z", "tenant_id":
 "3c003f65d8d64914a053f178fbbf953c", "os-extended-volumes:volumes_attached": [],
 "metadata": ...}}
...output omitted...
```

The deployment keeps looping trying to call the Compute service.

1.4. Cancel the deployment using **Ctrl+C**, and then delete the instance.

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

2. Determine the cause of the deployment failure

2.1. As the first step, check the `nova_conductor` service. Log in to `controller0` then become root.

```
[student@workstation ~(developer1-finance)]$ ssh controller0
[heat-admin@controller0 ~]$ sudo -i
[root@controller0 ~]#
```

2.2. Check the health of the `nova_conductor` container.

```
[root@controller0 ~]# docker exec -ti nova_conductor /openstack/healthcheck
Error response from daemon: Container
 8ffb511c9318d8024378f0419d709902a7601c33e59233203e7b7037c20c263b is restarting,
 wait until the container is running
```

2.3. Determine why `nova_conductor` is failing.

```
[root@controller0 ~]# docker logs nova_conductor
...output omitted...
++ cat /run_command
+ CMD='/usr/bin/nova-conductor '
+ ARGS=
+ [[ ! -n '' ]]
+ . kolla_extend_start
++ [[ ! -d /var/log/kolla/nova ]]
+++ stat -c %a /var/log/kolla/nova
++ [[ 2755 != \7\5\5 ]]
++ chmod 755 /var/log/kolla/nova
++ . /usr/local/bin/kolla_nova_extend_start
+ echo 'Running command: '\''/usr/bin/nova-conductor '\'''
+ exec /usr/bin/nova-conductor
Running command: '/usr/bin/nova-conductor '
Traceback (most recent call last):
  File "/usr/bin/nova-conductor", line 10, in <module>
    sys.exit(main())
  File "/usr/lib/python2.7/site-packages/nova/cmd/conductor.py", line 35, in main
    config.parse_args(sys.argv)
  File "/usr/lib/python2.7/site-packages/nova/config.py", line 52, in parse_args
    default_config_files=default_config_files)
  File "/usr/lib/python2.7/site-packages/oslo_config/cfg.py", line 2507, in
 __call__
    self._namespace._files_permission_denied)
oslo_config.cfg.ConfigFilesPermissionDeniedError: Failed to open some config
 files: /etc/nova/nova.conf
```

The error shows that the permissions on **`/etc/nova/nova.conf`** did not allow access for the user the container is running as.

2.4. Use the **`docker inspect`** command to determine the configuration file location for the **`nova_conductor`** service.

```
[root@controller0 ~]# docker inspect nova_conductor
...output omitted...
        "HostConfig": {
            "Binds": [
```

```
                "/var/lib/config-data/puppet-generated/nova/:/var/lib/kolla/
config_files/src:ro",
...output omitted...
```

2.5. The **/etc/nova/nova.conf** file shown in the error will be located at **/var/lib/config-data/puppet-generated/nova/etc/nova/nova.conf**. Check the permissions on the file then correct them.

```
[root@controller0 ~]# cd /var/lib/config-data/puppet-generated/nova
[root@controller0 nova]# ls -l etc/nova/nova.conf
-rw-------. 1 root 42436 60203 Nov  2 06:46 etc/nova/nova.conf
[root@controller0 nova]# chmod 640 etc/nova/nova.conf
[root@controller0 nova]# ls -l etc/nova/nova.conf
-rw-r-----. 1 root 42436 60203 Nov  2 06:46 etc/nova/nova.conf
```

2.6. Restart the `nova_conductor` container, wait for it to become healthy, and then log out of `controller0`.

```
[root@controller0 nova]# docker restart nova_conductor
nova_conductor
[root@controller0 nova]# docker ps | grep nova_conductor
8ffb511c9318  172.25.249.200:8787/rhosp13/openstack-nova-conductor:latest
 "kolla_start"  2 weeks ago  Up About a minute (healthy)
[root@controller0 nova]# logout
[heat-admin@controller0 ~]$ logout
[student@workstation ~(developer1-finance)]$
```

3. Retry launching the `finance-server1` instance.

```
[student@workstation ~(developer1-finance)]$ openstack server create \
--flavor default \
--image rhel7 \
--key-name example-keypair \
--network provider1-104 \
--wait \
finance-server1
...output omitted...
```

The deployment should succeed.

4. From `finance-server1`, ping `utility` at 10.0.104.1.

4.1. Obtain the console URL for `finance-server1`.

```
[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=a4d2bbb0-c625     |
|       | -4b38-acdb-0b5a3ffd907a                                         |
```

```
+-------+-------------------------------------------------------------+
```

4.2. Open the console URL in a web browser, and log in as `root` using `redhat` as the password. Use the **`ping`** command to test the connection to `utility` at 10.0.104.1.

```
[root@finance-server1 ~]# ping -c3 10.0.104.1
PING 10.0.104.1 (10.0.104.1) 56(84) bytes of data.
From 10.0.104.105 icmp_seq=1 Destination Host Unreachable
From 10.0.104.105 icmp_seq=2 Destination Host Unreachable
From 10.0.104.105 icmp_seq=3 Destination Host Unreachable

--- 10.0.104.1 ping statistics ---
3 packets transmitted, 0 received, +3 errors, 100% packet loss, time 2001ms
```

5. Investigate the communication problem to find the cause.

5.1. On `workstation`, determine the ID for the `provider1-104` network. This will be used to determine packet flow.

```
[student@workstation ~(developer1-finance)]$ openstack network show \
provider1-104 -c id
+-------+--------------------------------------+
| Field | Value                                |
+-------+--------------------------------------+
| id    | b9189edb-fc58-45fe-8eef-79f118ea2539 |
+-------+--------------------------------------+
```

5.2. Determine which compute node is hosting the `finance-server1` instance. Note also the generic instance name.

```
[student@workstation ~(developer1-finance)]$ source ~/architect1-finance-rc
[student@workstation ~(architect1-finance)]$ openstack server show finance-server1
+-------------------------------------+---------------------------------------+
| Field                               | Value                                 |
+-------------------------------------+---------------------------------------+
| OS-DCF:diskConfig                   | MANUAL                                |
| OS-EXT-AZ:availability_zone         | nova                                  |
| OS-EXT-SRV-ATTR:host                | compute0.overcloud.example.com        |
| OS-EXT-SRV-ATTR:hypervisor_hostname | compute0.overcloud.example.com        |
| OS-EXT-SRV-ATTR:instance_name       | instance-0000000a                     |
| OS-EXT-STS:power_state              | Running                               |
...output omitted...
```

5.3. Log in to the appropriate compute node, and change to the `root` user.

```
[student@workstation ~(architect1-finance)]$ ssh compute0
[heat-admin@compute0 ~]$ sudo -i
[root@compute0 ~]#
```

5.4. Find the TAP device used by the instance. Note that it should be connected to `br-int`. Also note the MAC address; this will be used when tracing packet flow.

```
[root@compute0 ~]# virsh dumpxml instance-0000000a
...output omitted...
    <interface type='bridge'>
```

```
      <mac address='fa:16:3e:05:e0:c2'/>
      <source bridge='br-int'/>
      <virtualport type='openvswitch'>
        <parameters interfaceid='fc742bd3-2133-4a29-9a53-d4cc56e61a1f'/>
      </virtualport>
      <target dev='tapfc742bd3-21'/>
      <model type='virtio'/>
      <mtu size='1500'/>
      <alias name='net0'/>
      <address type='pci' domain='0x0000' bus='0x00' slot='0x03' function='0x0'/>
    </interface>
...output omitted...
```

5.5. View the OVS connections for the `provider1-104` network and `finance-server1` TAP device.

```
[root@compute0 ~]# ovs-vsctl show
...output omitted...
    Bridge "br-eth3"
        fail_mode: standalone
        Port "br-eth3"
            Interface "br-eth3"
                type: internal
        Port "patch-provnet-b9189edb-fc58-45fe-8eef-79f118ea2539-to-br-int"
            Interface "patch-provnet-b9189edb-fc58-45fe-8eef-79f118ea2539-to-br-
int"
                type: patch
                options: {peer="patch-br-int-to-provnet-b9189edb-
fc58-45fe-8eef-79f118ea2539"}
    Bridge br-int
        fail_mode: secure
...output omitted...
        Port "patch-br-int-to-provnet-b9189edb-fc58-45fe-8eef-79f118ea2539"
            Interface "patch-br-int-to-provnet-b9189edb-
fc58-45fe-8eef-79f118ea2539"
                type: patch
                options: {peer="patch-provnet-b9189edb-
fc58-45fe-8eef-79f118ea2539-to-br-int"}
        Port "tapfc742bd3-21"
            Interface "tapfc742bd3-21"
```

Note that the TAP device connects the instance to `br-int`. Also note that the creation of the `provider1-104` network resulted in a pair of interfaces connecting `br-int` to `br-eth3` with a patch port. The interface names contain the ID of the `provider1-104` network.

5.6. Tracing packets through OpenFlow rules is tedious and error-prone. Use the **`ovs-appctl`** command to simulate sending an ICMP packet from `finance-server1` to `utility`. Given the OVS connections shown in the previous step, traffic from the

instance should enter `br-int` through the `tapfc742bd3-21` port, then travel over the patch port to `br-eth3`.

The table below describes the packet fields used in the command.

Packet Fields

FIELD	DESCRIPTION
in_port=	Defines the port the packet is entering the switch on. The TAP port of `finance-server1` is `tapfc742bd3-21`.
protocol	The protocol of the packet. This example is simulating a ping so the protocol is ICMP.
nw_src=	The source network (IP) address. The IP address of `finance-server1` is 10.0.104.105.
dl_src=	The ethernet address (MAC) of the source machine. The MAC address of `finance-server1` is fa:16:3e:05:e0:c2.
nw_dst=	The destination network (IP) address. The IP address of `utility` is 10.0.104.1.

NOTE
The command input below has been broken across lines for readability; do not include carriage returns when you enter this command.

```
[root@compute0 ~]# ovs-appctl ofproto/trace br-int \
in_port="tapfc742bd3-21",icmp,nw_src=10.0.104.105,
dl_src=fa:16:3e:05:e0:c2,nw_dst=10.0.104.1
Flow:
 icmp,in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:05:e0:c2,dl_dst=00:00:00:00:00:00,
nw_src=10.0.104.105,nw_dst=10.0.104.1,nw_tos=0,nw_ecn=0,nw_ttl=0,icmp_type=0,
icmp_code=0

bridge("br-int")
----------------
 0. in_port=4, priority 100
    set_field:0x1->reg13
    set_field:0x3->reg11
    set_field:0x2->reg12
    set_field:0x8->metadata
    set_field:0x4->reg14
    resubmit(,8)
 8. reg14=0x4,metadata=0x8,dl_src=fa:16:3e:05:e0:c2, priority 50, cookie
 0xab405c15
    resubmit(,9)
 9. ip,reg14=0x4,metadata=0x8,dl_src=fa:16:3e:05:e0:c2,nw_src=10.0.104.105,
 priority 90, cookie 0x6d4e4677
    resubmit(,10)
10. metadata=0x8, priority 0, cookie 0xb42ed0a4
    resubmit(,11)
11. ip,metadata=0x8, priority 100, cookie 0xa4029170
```

```
    load:0x1->NXM_NX_XXREG0[96]
    resubmit(,12)
12. metadata=0x8, priority 0, cookie 0x6603eb2c
    resubmit(,13)
13. ip,reg0=0x1/0x1,metadata=0x8, priority 100, cookie 0xa00b21a5
    ct(table=14,zone=NXM_NX_REG13[0..15])
    drop
     -> A clone of the packet is forked to recirculate. The forked pipeline will
 be resumed at table 14.

Final flow: icmp,reg0=0x1,reg11=0x3,reg12=0x2,reg13=0x1,reg14=0x4,metadata=0x8,
in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:05:e0:c2,dl_dst=00:00:00:00:00:00,
nw_src=10.0.104.105,nw_dst=10.0.104.1,nw_tos=0,nw_ecn=0,nw_ttl=0,icmp_type=0,i
icmp_code=0
Megaflow:
 recirc_id=0,eth,ip,in_port=4,vlan_tci=0x0000/0x1000,dl_src=fa:16:3e:05:e0:c2,
nw_src=10.0.104.105,nw_dst=0.0.0.0/1,nw_frag=no
Datapath actions: ct(zone=1),recirc(0xa)

===============================================================================
recirc(0xa) - resume conntrack with default ct_state=trk|new (use --ct-next to
 customize)
===============================================================================

Flow: recirc_id=0xa,ct_state=new|
trk,ct_zone=1,eth,icmp,reg0=0x1,reg11=0x3,reg12=0x2,
reg13=0x1,reg14=0x4,metadata=0x8,in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:05:e0:c2,
dl_dst=00:00:00:00:00:00,nw_src=10.0.104.105,nw_dst=10.0.104.1,nw_tos=0,nw_ecn=0,
nw_ttl=0,icmp_type=0,icmp_code=0

bridge("br-int")
----------------
    thaw
        Resuming from table 14
14. ct_state=+new-est+trk,ip,reg14=0x4,metadata=0x8, priority 2002, cookie
 0x43cfb2b5
    load:0x1->NXM_NX_XXREG0[97]
    resubmit(,15)
15. metadata=0x8, priority 0, cookie 0xba3f581d
    resubmit(,16)
16. metadata=0x8, priority 0, cookie 0x42968196
    resubmit(,17)
17. metadata=0x8, priority 0, cookie 0x8d3beedd
    resubmit(,18)
18. ip,reg0=0x2/0x2,metadata=0x8, priority 100, cookie 0x8c9c9578
    ct(commit,zone=NXM_NX_REG13[0..15],exec(load:0->NXM_NX_CT_LABEL[0]))
    load:0->NXM_NX_CT_LABEL[0]
    resubmit(,19)
19. metadata=0x8, priority 0, cookie 0x491f8315
    resubmit(,20)
20. metadata=0x8, priority 0, cookie 0x66f27962
    resubmit(,21)
21. metadata=0x8, priority 0, cookie 0xc32c3d73
    resubmit(,22)
22. metadata=0x8, priority 0, cookie 0xdebd3512
    resubmit(,23)
```

```
 23. metadata=0x8, priority 0, cookie 0x8f98275c
     resubmit(,24)
 24. metadata=0x8, priority 0, cookie 0x6f85b1ed
     set_field:0xfffe->reg15
     resubmit(,32)
 32. priority 0
     resubmit(,33)
 33. reg15=0xfffe,metadata=0x8, priority 100
     set_field:0x4->reg13
     set_field:0x1->reg15
     resubmit(,34)
     34. priority 0
             set_field:0->reg0
             set_field:0->reg1
             set_field:0->reg2
             set_field:0->reg3
             set_field:0->reg4
             set_field:0->reg5
             set_field:0->reg6
             set_field:0->reg7
             set_field:0->reg8
             set_field:0->reg9
             resubmit(,40)
         40. metadata=0x8, priority 0, cookie 0xb484f3de
             resubmit(,41)
         41. ip,reg15=0x1,metadata=0x8, priority 110, cookie 0x1d2d7934
             resubmit(,42)
         42. metadata=0x8, priority 0, cookie 0xd311b4f8
             resubmit(,43)
         43. metadata=0x8, priority 0, cookie 0x9ec6220a
             resubmit(,44)
         44. metadata=0x8, priority 0, cookie 0xe1453aba
             resubmit(,45)
         45. metadata=0x8, priority 0, cookie 0x3fa4aed4
             resubmit(,46)
         46. metadata=0x8, priority 0, cookie 0x61ba3940
             resubmit(,47)
         47. metadata=0x8, priority 0, cookie 0x6efeaece
             resubmit(,48)
         48. metadata=0x8, priority 0, cookie 0x1734102d
             resubmit(,49)
         49. reg15=0x1,metadata=0x8, priority 50, cookie 0x75acab86
             resubmit(,64)
         64. priority 0
             resubmit(,65)
         65. reg15=0x1,metadata=0x8, priority 100
             push_vlan:0x8100
             set_field:4200->vlan_vid
             output:5

             bridge("br-eth3")
             -----------------
                  0. priority 0
                     NORMAL
                      -> no learned MAC for destination, flooding
             pop_vlan
```

```
    set_field:0xfffe->reg15

Final flow: recirc_id=0xa,eth,icmp,reg11=0x3,reg12=0x2,reg13=0x4,reg14=0x4,
reg15=0xfffe,metadata=0x8,in_port=4,vlan_tci=0x0000,dl_src=fa:16:3e:05:e0:c2,
dl_dst=00:00:00:00:00:00,nw_src=10.0.104.105,nw_dst=10.0.104.1,nw_tos=0,nw_ecn=0,
nw_ttl=0,icmp_type=0,icmp_code=0
Megaflow: recirc_id=0xa,ct_state=+new-est-rel-rpl-inv+trk,ct_label=0/0x1,eth,icmp,
in_port=4,vlan_tci=0x0000/0x1fff,dl_src=fa:16:3e:05:e0:c2,dl_dst=00:00:00:00:00:00,
nw_src=10.0.104.105,nw_dst=10.0.104.1,nw_frag=no
Datapath actions: ct(commit,zone=1,label=0/0x1),push_vlan(vid=104,pcp=0),3
```

The simulated ICMP traffic does reach `br-eth3` successfully.

5.7. Review the configuration of `br-eth3`.

```
[root@compute0 ~]# ovs-vsctl show
...output omitted...
    Bridge "br-eth3"
        fail_mode: standalone
        Port "br-eth3"
            Interface "br-eth3"
                type: internal
        Port "patch-provnet-b9189edb-fc58-45fe-8eef-79f118ea2539-to-br-int"
            Interface "patch-provnet-b9189edb-fc58-45fe-8eef-79f118ea2539-to-br-
int"
                type: patch
                options: {peer="patch-br-int-to-provnet-b9189edb-
fc58-45fe-8eef-79f118ea2539"}
...output omitted...
```

Note that there are only an internal and a patch port present; there is no external interface shown.

5.8. View the IP configuration for the compute node.

```
[root@compute0 ~]# ip a
...output omitted...
5: eth3: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen
 1000
    link/ether 52:54:00:03:00:02 brd ff:ff:ff:ff:ff:ff
6: eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master ovs-
system 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
7: ovs-system: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default
 qlen 1000
    link/ether 86:46:ff:6a:61:a8 brd ff:ff:ff:ff:ff:ff
8: genev_sys_6081: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 65000 qdisc noqueue
 master ovs-system state UNKNOWN group default qlen 1000
    link/ether 46:3f:1b:5a:6d:29 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::443f:1bff:fe5a:6d29/64 scope link
       valid_lft forever preferred_lft forever
9: br-int: <BROADCAST,MULTICAST> mtu 1500 qdisc noop state DOWN group default qlen
 1000
    link/ether 86:a7:a5:b8:33:42 brd ff:ff:ff:ff:ff:ff
```

```
10: br-eth3: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UNKNOWN group default qlen 1000
    link/ether 52:54:00:03:00:02 brd ff:ff:ff:ff:ff:ff
    inet6 fe80::5054:ff:fe03:2/64 scope link
       valid_lft forever preferred_lft forever
11: br-eth4: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state
 UNKNOWN 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...
```

The `eth3` interface is listed as **down**.

5.9. Review the **`/etc/sysconfig/network-scripts/ifcfg-eth3`** file.

```
[root@compute0 ~]# cat /etc/sysconfig/network-scripts/ifcfg-eth3
# This file is autogenerated by os-net-config
DEVICE=eth3
ONBOOT=no
HOTPLUG=no
NM_CONTROLLED=no
PEERDNS=no
DEVICETYPE=ovs
TYPE=OVSPort
OVS_BRIDGE=br-eth3
BOOTPROTO=none
```

The interface has been disabled at boot.

5.10. Bring up the `eth3` interface.

```
[root@compute0 ~]# ifup eth3
```

5.11. Review the configuration of `br-eth3` again.

```
[root@compute0 ~]# ovs-vsctl show
...output omitted...
    Bridge "br-eth3"
        fail_mode: standalone
        Port "eth3"
            Interface "eth3"
        Port "br-eth3"
            Interface "br-eth3"
                type: internal
...output omitted...
```

The `eth3` interface is now shown in the output.

5.12. Change the **`ONBOOT`** option to **`yes`** for `eth3`.

```
# This file is autogenerated by os-net-config
DEVICE=eth3
ONBOOT=yes
...output omitted...
```

6. Use the **ping** command to retest communication between `finance-server1` and `utility`.

```
[root@finance-server1 ~]# ping -c3 10.0.104.1
PING 10.0.104.1 (10.0.104.1) 56(84) bytes of data.
64 bytes from 10.0.104.1: icmp_seq=1 ttl=64 time=0.193 ms
64 bytes from 10.0.104.1: icmp_seq=2 ttl=64 time=0.228 ms
64 bytes from 10.0.104.1: icmp_seq=3 ttl=64 time=0.191 ms

--- 10.0.104.1 ping statistics ---
3 packets transmitted, 3 received, 0% packet loss, time 1999ms
rtt min/avg/max/mdev = 0.191/0.204/0.228/0.017 ms
```

Cleanup

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

```
[student@workstation ~]$ lab troubleshooting-services cleanup
```

This concludes the guided exercise.

TROUBLESHOOTING OPENSTACK

PERFORMANCE CHECKLIST

In this lab, you will find and fix issues in the OpenStack environment. You will solve problems in the areas of authentication, networking, compute nodes, and security. Finally, you will launch an instance and ensure that everything is working as it should.

OUTCOMES

You should be able to:

- Troubleshoot authentication issues within OpenStack
- Search log files to help describe the nature of the problem
- Troubleshoot messaging issues within OpenStack
- Troubleshoot networking issues within OpenStack

BEFORE YOU BEGIN

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

From `workstation`, run **`lab troubleshooting-review setup`**, which verifies that OpenStack services are running and the resources required for the lab are available. This script also breaks the Compute configuration, authentication, and networking.

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

1. As the `operator1` user in the production project, remove the existing image named `production-rhel7`. Troubleshoot any problems.
2. Source the **`/home/student/architect1-production-rc`** environment file. Run the **`lab troubleshooting-review break`** command to set up the next part of the lab exercise.

```
[student@workstation ~(operator1-production)]$ source ~/architect1-production-rc
[student@workstation ~(architect1-production)]$ lab troubleshooting-review break
```

3. Source the **`/home/student/operator1-production-rc`** environment file, then list all OpenStack images. Troubleshoot and fix any issues.
4. Create a new server instance named `production-web1` using the settings from the table below. This action will fail. Troubleshoot and fix any issues.

Instance Options

OPTION	VALUE
`--flavor`	default
`--image`	rhel7

OPTION	VALUE
`--key-name`	example-keypair
`--network`	production-network1
`--security-group`	production-web

5. Create a floating IP address and assign it to the instance. Troubleshoot and fix any issues.
6. Attempt to log in to `production-web1` using SSH. Troubleshoot and fix any issues.
7. Create a 1 GB volume named `production-volume1`. Verify the volume status. Use the admin user's environment file on `controller0` at **`/home/heat-admin/overcloudrc`**. Troubleshoot and fix any issues.

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab troubleshooting-review cleanup
```

This concludes the lab.

TROUBLESHOOTING OPENSTACK

PERFORMANCE CHECKLIST

In this lab, you will find and fix issues in the OpenStack environment. You will solve problems in the areas of authentication, networking, compute nodes, and security. Finally, you will launch an instance and ensure that everything is working as it should.

OUTCOMES

You should be able to:

- Troubleshoot authentication issues within OpenStack
- Search log files to help describe the nature of the problem
- Troubleshoot messaging issues within OpenStack
- Troubleshoot networking issues within OpenStack

BEFORE YOU BEGIN

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

From `workstation`, run **`lab troubleshooting-review setup`**, which verifies that OpenStack services are running and the resources required for the lab are available. This script also breaks the Compute configuration, authentication, and networking.

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

1. As the `operator1` user in the production project, remove the existing image named `production-rhel7`. Troubleshoot any problems.

   1.1. Source the **`/home/student/operator1-production-rc`** file.

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

   1.2. Delete the `production-rhel7` image.

```
[student@workstation ~(operator1-production)]$ openstack image delete \
production-rhel7
Failed to delete image with name or ID '21b3b8ba-e28e-4b41-9150-ac5b44f9d8ef': 403
 Forbidden
Image 21b3b8ba-e28e-4b41-9150-ac5b44f9d8ef is protected and cannot be deleted.
    (HTTP 403)
    Failed to delete 1 of 1 images.
```

   1.3. The error you see is because the image is currently protected. You need to unprotect the image before it can be deleted.

```
[student@workstation ~(operator1-production)]$ openstack image set \
```

```
--unprotected production-rhel7
[student@workstation ~(operator1-production)]$ openstack image delete \
production-rhel7
```

2. Source the **/home/student/architect1-production-rc** environment file. Run the **lab troubleshooting-review break** command to set up the next part of the lab exercise.

```
[student@workstation ~(operator1-production)]$ source ~/architect1-production-rc
[student@workstation ~(architect1-production)]$ lab troubleshooting-review break
```

3. Source the **/home/student/operator1-production-rc** environment file, then list all OpenStack images. Troubleshoot and fix any issues.

   3.1. List all the OpenStack images.

```
[student@workstation ~(architect1-production)]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack image list
Failed to discover available identity versions when contacting
 http://172.25.251.50:5000/v3. Attempting to parse version from URL.
Unable to establish connection to http://172.25.251.50:5000/v3/
auth/tokens: HTTPConnectionPool(host='172.25.251.50', port=5000):
 Max retries exceeded with url: /v3/auth/tokens (Caused by
 NewConnectionError('<requests.packages.urllib3.connection.HTTPConnection object
 at 0x7fbbbbc445d0>: Failed to establish a new connection: [Errno 111] Connection
 refused',))
```

   3.2. The error occurs because OpenStack cannot authenticate the `operator1` user. This can happen when the environment file for the user contains incorrect information. Log in to `controller0`. Search for **listen keystone_public** in **/var/lib/config-data/puppet-generated/haproxy/ etc/haproxy/haproxy.cfg**. The second IP address is the one that must be used in the environment file. When done, log off from the controller node.

```
[student@workstation ~(operator1-production)]$ ssh heat-admin@controller0
[heat-admin@controller0 ~]$ sudo cat \
/var/lib/config-data/puppet-generated/haproxy/etc/haproxy/haproxy.cfg
...output omitted...
listen keystone_public
 bind 172.24.1.50:5000 transparent
 bind 172.25.250.50:5000 transparent
...output omitted...
[heat-admin@controller0 ~]$ logout
[student@workstation ~(operator1-production)]$
```

   3.3. Compare the IP address from HAProxy with the OS_AUTH_URL variable in the environment file. Update the environment file to match if required.

```
...output omitted...
export OS_AUTH_URL=http://172.25.250.50:5000/v3
```

```
...output omitted...
```

3.4. Source the **/home/student/operator1-production-rc** file again. Use the **openstack image list** command to ensure that the OS_AUTH_URL option is correct.

```
[student@workstation ~(operator1-production)]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack image list
+--------------------------------------+-----------------+--------+
| ID                                   | Name            | Status |
+--------------------------------------+-----------------+--------+
| ec9473de-4048-4ebb-b08a-a9be619477ac | octavia-amphora | active |
| 6b0128a9-4481-4ceb-b34e-ffe92e0dcfdd | rhel7           | active |
| 5f7f8208-33b5-4f17-8297-588f938182c0 | rhel7-db        | active |
| 14b7e8b2-7c6d-4bcf-b159-1e4e7582107c | rhel7-web       | active |
+--------------------------------------+-----------------+--------+
```

4. Create a new server instance named `production-web1` using the settings from the table below. This action will fail. Troubleshoot and fix any issues.

Instance Options

OPTION	VALUE
--flavor	default
--image	rhel7
--key-name	example-keypair
--network	production-network1
--security-group	production-web

4.1. Create a new server instance.

```
[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=production-network1 \
--security-group production-web \
--image rhel7 --wait production-web1
Error creating server: production-web1
Error creating server
```

4.2. This error is due to a problem with the Compute service. List the Compute services. Source the **/home/student/architect1-production-rc** environment file first, because `operator1` does not have permission to interact directly with the Compute services.

```
[student@workstation ~(operator1-production)]$ source ~/architect1-production-rc
[student@workstation ~(architect1-production)]$ openstack compute service list \
-c Binary -c Host -c Status
+------------------+----------------------------------+----------+
| Binary           | Host                             | Status   |
```

```
+------------------+----------------------------------+----------+
| nova-conductor   | controller0.overcloud.example.com | enabled  |
| nova-compute     | compute1.overcloud.example.com    | disabled |
| nova-compute     | computehci0.overcloud.example.com | enabled  |
| nova-compute     | compute0.overcloud.example.com    | disabled |
| nova-consoleauth | controller0.overcloud.example.com | enabled  |
| nova-scheduler   | controller0.overcloud.example.com | enabled  |
+------------------+----------------------------------+----------+
```

4.3. Enable the `nova-compute` services on `compute0` and `compute1`.

```
[student@workstation ~(architect1-production)]$ openstack compute service set \
--enable compute0.overcloud.example.com nova-compute
[student@workstation ~(architect1-production)]$ openstack compute service set \
--enable compute1.overcloud.example.com nova-compute
```

4.4. Verify the state of the `nova-compute` services.

```
[student@workstation ~(architect1-production)]$ openstack compute service list \
-c Binary -c Host -c Status
+------------------+----------------------------------+---------+
| Binary           | Host                             | Status  |
+------------------+----------------------------------+---------+
| nova-conductor   | controller0.overcloud.example.com | enabled |
| nova-compute     | compute1.overcloud.example.com    | enabled |
| nova-compute     | computehci0.overcloud.example.com | enabled |
| nova-compute     | compute0.overcloud.example.com    | enabled |
| nova-consoleauth | controller0.overcloud.example.com | enabled |
| nova-scheduler   | controller0.overcloud.example.com | enabled |
+------------------+----------------------------------+---------+
```

4.5. Source the **`/home/student/operator1-production-rc`** environment file, delete the failed instance, and then try to create the instance again.

```
[student@workstation ~(architect1-production)]$ source ~/operator1-production-rc
[student@workstation ~(operator1-production)]$ openstack server delete \
production-web1
[student@workstation ~(operator1-production)]$ openstack server list

[student@workstation ~(operator1-production)]$ openstack server create \
--flavor default \
--key-name example-keypair \
--nic net-id=production-network1 \
--security-group production-web \
--image rhel7 --wait production-web1
```

5. Create a floating IP address and assign it to the instance. Troubleshoot and fix any issues.

5.1. Create a floating IP address.

```
[student@workstation ~(operator1-production)]$ openstack floating ip create \
provider-datacentre
+---------------------+---------------------+------------------+------+
| ID                  | Floating IP Address | Fixed IP Address | Port |
+---------------------+---------------------+------------------+------+
```

```
| ce31(...)9ecb      | 172.25.250.N        | None              | None |
+--------------------+---------------------+-------------------+------+
[student@workstation ~(operator1-production)]$ openstack server add \
floating ip production-web1 172.25.250.N
Unable to associate floating IP 172.25.250.N to fixed IP 192.168.0.6
 for instance a53e66d9-6413-4ae4-b95b-2012dd52f908. Error: External
 network 7aaf57c1-3c34-45df-94d3-dbc12754b22e is not reachable from subnet
 cfc7ddfa-4403-41a7-878f-e8679596eafd.  Therefore, cannot associate Port
 dcb6692d-0094-42ec-bc8e-a52fd97d7a4c with a Floating IP.
Neutron server returns request_ids: ['req-4f88fb24-7628-4155-a921-ff628cb4b371']
 (HTTP 400) (Request-ID: req-d6862c58-66c4-44b6-a4d1-bf26514bf04b)
```

This error message occurs because the external network is not attached to the router that is the gateway for the `production-network1` network.

5.2. Connect `production-router1` to the `provider-datacentre` external network.

```
[student@workstation ~(operator1-production)]$ openstack router set \
--external-gateway provider-datacentre production-router1
```

5.3. Attach the floating IP address to the instance. Verify that the instance has been assigned the floating IP address.

```
[student@workstation ~(operator1-production)]$ openstack server add \
floating ip production-web1 172.25.250.N
[student@workstation ~(operator1-production)]$ openstack server list \
-c Name -c Networks
+-----------------+-----------------------------------------------+
| Name            | Networks                                      |
+-----------------+-----------------------------------------------+
| production-web1 | production-network1=192.168.0.P, 172.25.250.N |
+-----------------+-----------------------------------------------+
```

6. Attempt to log in to `production-web1` using SSH. Troubleshoot and fix any issues.

6.1. Attempt to log in to the instance using SSH.

```
[student@workstation ~(operator1-production)]$ ssh cloud-user@172.25.250.N
ssh: connect to host 172.25.250.N port 22: Connection timed out
```

6.2. Find out which security group the instance is using, and then list the rules in that security group.

```
[student@workstation ~(operator1-production)]$ openstack server show \
production-web1 -f json
...output omitted...
  "security_groups": "name='production-web'",
...output omitted...
[student@workstation ~(operator1-production)]$ openstack security group rule \
list production-web
+---------------+-------------+----------+------------+-----------------------+
| ID            | IP Protocol | IP Range | Port Range | Remote Security Group |
+---------------+-------------+----------+------------+-----------------------+
| cc92(...)95b1 | None        | None     |            | None                  |
| eb84(...)c6e7 | None        | None     |            | None                  |
```

```
+---------------+-------------+----------+------------+-----------------------+
```

Note that there is no rule to allow SSH access to the instance.

6.3. Create a security group rule to allow SSH access.

```
[student@workstation ~(operator1-production)]$ openstack security group rule \
create --protocol tcp --dst-port 22:22 production-web
+-------------------+--------------------------------------+
| Field             | Value                                |
+-------------------+--------------------------------------+
| created_at        | 2017-06-12T07:24:34Z                 |
| description       |                                      |
| direction         | ingress                              |
| ethertype         | IPv4                                 |
| headers           |                                      |
| id                | 06070264-1427-4679-bd8e-e3a8f2e189e9 |
| port_range_max    | 22                                   |
| port_range_min    | 22                                   |
| project_id        | 9913a8abd192443c96587a8dc1c0a364     |
| project_id        | 9913a8abd192443c96587a8dc1c0a364     |
| protocol          | tcp                                  |
| remote_group_id   | None                                 |
| remote_ip_prefix  | 0.0.0.0/0                            |
| revision_number   | 1                                    |
| security_group_id | ac9ae6e6-0056-4501-afea-f83087b8297f |
| updated_at        | 2017-06-12T07:24:34Z                 |
+-------------------+--------------------------------------+
```

6.4. Attempt to log in to the instance again.

```
[student@workstation ~(operator1-production)]$ ssh cloud-user@172.25.250.N
Warning: Permanently added '172.25.250.N' (ECDSA) to the list of known hosts.
[cloud-user@production-web1 ~]$ logout
[student@workstation ~(operator1-production)]$
```

7. Create a 1 GB volume named `production-volume1`. Verify the volume status. Use the admin user's environment file on `controller0` at **`/home/heat-admin/overcloudrc`**. Troubleshoot and fix any issues.

7.1. Create the volume.

```
[student@workstation ~(operator1-production)]$ openstack volume create \
--size 1 production-volume1
...output omitted...
```

7.2. Determine the status of `production-volume1`.

```
[student@workstation ~(operator1-production)]$ openstack volume list
+---------------+--------------------+-----------+------+-------------+
| ID            | Display Name       | Status    | Size | Attached to |
+---------------+--------------------+-----------+------+-------------+
| 0da8(...)be3f | production-volume1 | creating  |    1 |             |
```

```
+---------------+--------------------+-----------+------+-------------+
```

The volume status stays at **creating**.

7.3. Log in to `controller0` as `heat-admin`, and then review the log files for the Block Storage Scheduler service.

```
[student@workstation ~(operator1-production)]$ ssh heat-admin@controller0
[heat-admin@controller ~]$ sudo less \
/var/log/containers/cinder/cinder-scheduler.log
...output omitted...
2018-11-16 01:57:52.112 1 ERROR oslo_service.service MessageDeliveryFailure:
 Unable to connect to AMQP server on controller0.
internalapi.overcloud.example.com:5672 after None tries: (0, 0): (403)
 ACCESS_REFUSED - Login was refused using authentication mechanism AMQPLAIN. For
 details see the broker logfile.
...output omitted...
```

The repeated error above indicates that the Block Storage Scheduler service is unable to authenticate to RabbitMQ.

7.4. Inspect the Block Storage Volume service log file for errors.

```
(overcloud) [heat-admin@controller0 ~]$ sudo less \
/var/log/containers/cinder/cinder-volume.log
...output omitted...
2018-11-16 01:00:47.170 194544 ERROR oslo_service.service
 [req-2086d653-8982-40dc-9e75-7c0bba9e4774 - - - - -] Error starting
 thread.: MessageDeliveryFailure: Unable to connect to AMQP server on
 controller0.internalapi.overcloud.example.com:5672 after None tries: (0,
 0): (403) ACCESS_REFUSED - Login was refused using authentication mechanism
 AMQPLAIN. For details see the broker logfile.
...output omitted...
```

The error above indicates that the Volume service is also unable to authenticate to RabbitMQ.

7.5. Source the **/home/heat-admin/overcloudrc** environment file, and then determine the state of the Volume services.

```
[heat-admin@controller0 ~]$ source ~/overcloudrc
(overcloud) [heat-admin@controller0 ~]$ openstack volume service list \
-c Binary -c Host -c Status -c State
+------------------+------------------------+---------+-------+
| Binary           | Host                   | Status  | State |
+------------------+------------------------+---------+-------+
| cinder-scheduler | controller0            | enabled | down  |
| cinder-volume    | hostgroup@tripleo_ceph | enabled | down  |
+------------------+------------------------+---------+-------+
```

7.6. Use the **docker exec** command to list the RabbitMQ users.

```
(overcloud) [heat-admin@controller0 ~]$ sudo docker exec -t \
rabbitmq-bundle-docker-0 rabbitmqctl list_users
Listing users
```

```
guest [administrator]
```

The only nonadministrative user is `guest`

7.7. Investigate the block service configuration file. Locate the **`transport_url`** option.

```
(overcloud) [heat-admin@controller0 ~]$ sudo grep ^transport_url \
/var/lib/config-data/puppet-generated/cinder/etc/cinder/cinder.conf
transport_url=rabbit://change_me:V6VpJgBNvThefbnRyE3xx9sag@controller0.internalapi.
overcloud.example.com:5672/?ssl=0
transport_url=rabbit://change_me:V6VpJgBNvThefbnRyE3xx9sag@controller0.internalapi.
overcloud.example.com:5672/?ssl=0
```

The user portion of the URL is set to `change_me`, which does not exist as a RabbitMQ user.

7.8. Change the username in both **`transport_url`** options to `guest`. Restart the `openstack-cinder-volume` Pacemaker resource, and the `cinder_scheduler` containerized service.

```
(overcloud) [heat-admin@controller0 ~]$ sudo pcs resource restart \
openstack-cinder-volume
openstack-cinder-volume successfully restarted
(overcloud) [heat-admin@controller0 ~]$ sudo docker restart cinder_scheduler
cinder_scheduler
(overcloud) [heat-admin@controller0 ~]$ logout
[student@workstation ~(operator1-production)]$
```

7.9. On `workstation`, delete the incorrect volume and recreate it. Verify that it has been properly created.

```
[student@workstation ~(operator1-production)]$ openstack volume delete \
production-volume1
[student@workstation ~(operator1-production)]$ openstack volume create \
--size 1 production-volume1
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| attachments         | []                                   |
| availability_zone   | nova                                 |
| bootable            | false                                |
| consistencygroup_id | None                                 |
| created_at          | 2017-06-14T08:08:01.726844           |
| description         | None                                 |
| encrypted           | False                                |
| id                  | 128a9514-f8bd-4162-9f7e-72036f684cba |
| multiattach         | False                                |
| name                | production-volume1                   |
| properties          |                                      |
| replication_status  | disabled                             |
| size                | 1                                    |
| snapshot_id         | None                                 |
| source_volid        | None                                 |
| status              | creating                             |
| type                | None                                 |
| updated_at          | None                                 |
```

```
| user_id              | 0ac575bb96e24950a9551ac4cda082a4     |
+----------------------+--------------------------------------+
[student@workstation ~(operator1-production)]$ openstack volume list
+--------------------------------------+--------------------+-----------+------+
| ID                                   | Display Name       | Status    | Size |
+--------------------------------------+--------------------+-----------+------+
| 128a9514-f8bd-4162-9f7e-72036f684cba | production-volume1 | available |    1 |
+--------------------------------------+--------------------+-----------+------+
```

Evaluation

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

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

Cleanup

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

```
[student@workstation ~]$ lab troubleshooting-review cleanup
```

This concludes the lab.

SUMMARY

In this chapter, you learned:

- The **`openstack`** client command includes a **`--debug`** option, which allows you to see the request and response payloads. The output can also be logged to a file for later analysis using the **`--log-file`** option.
- Network issues may require troubleshooting at the physical layer or at the SDN layer. Common commands for network troubleshooting include **`ip link show`**, **`tcpdump`**, and **`ovs-vsctl`**.
- OVN implements all of its functions using OpenFlow rules. These rules can be difficult to navigate; the **`ovn-trace`** and **`ovs-appctl ofproto/trace`** commands allow administrators to trace the path for a packet with a given set of properties.
- The Compute scheduler component selects compute nodes based on an algorithm. By default, this algorithm passes the list of all nodes through a set of filters which removes invalid nodes, and then selects from the nodes that remain. The **`no valid host`** error means that the Compute scheduler has not identified a compute node that can provide the resources required by the instance.
- To access an instance using a floating IP address, both the external network associated with that floating IP address and the internal network to which the instance is connected have to be connected using a router.
- Images can be protected to prevent accidental deletion.
- To troubleshoot issues when launching instances, check that all the required user and project infrastructure, such as networks, routers, and key pairs, are in place first.
- The OpenStack block storage service requires that the `openstack` user has read, write, and execute capabilities in both the `volumes` and the `images` pool in Ceph.

CHAPTER 10

COMPREHENSIVE REVIEW

GOAL

Review tasks from *Red Hat OpenStack Administration II: Infrastructure Configuration for Cloud Administrators*

OBJECTIVES

Review tasks from *Red Hat OpenStack Administration II: Infrastructure Configuration for Cloud Administrators*

SECTIONS

- Comprehensive Review

LAB

- Lab: Building a Custom Image
- Lab: Managing Project Organization
- Lab: Launching an Application Stack

COMPREHENSIVE REVIEW

OBJECTIVES

After completing this section, students should have reviewed and refreshed the knowledge and skills learned in *Red Hat OpenStack Administration II: Infrastructure Configuration for Cloud Administrators*.

REVIEWING *RED HAT OPENSTACK ADMINISTRATION II: INFRASTRUCTURE CONFIGURATION FOR CLOUD ADMINISTRATORS*

Before beginning the comprehensive review for this course, students should be comfortable with the topics covered in each chapter.

Students can refer to earlier sections in the textbook for extra study.

Chapter 1, *Navigating the Red Hat OpenStack Platform Architecture*

Describe the undercloud and overcloud architecture and service components.

- Describe the classroom environment, support systems, and services.
- Describe the functions of the undercloud components and the management architecture.
- Describe containerized services and management commands.
- Describe the functions of the overcloud components and the management architecture.

Chapter 2, *Describing the OpenStack Control Plane*

Introduce the critical core services making up the OpenStack control plane.

- Describe service endpoint configuration and security.
- Describe message broker patterns and implementation.
- Identify the shared services running on a controller node.

Chapter 3, *Integrating Identity Management*

Configure a scalable identity service and manage scalable projects.

- Describe the installation and architecture of a Red Hat Identity Management back end for the OpenStack identity service.
- Manage user tokens that implement user authorization for accessing OpenStack services.
- Manage project quotas, domains, hierarchies, and groups.

Chapter 4, *Performing Image Operations*

Customize instance deployments by creating build-to-order images, modifying guest and disk images, and initializing instances when deployed.

- Describe common image formats, features, and use cases.

- Build an image using diskimage-builder.
- Modify existing images with additional software and configuration using **`guestfish`** and **`virt-customize`**.
- Customize instances during deployment using **`cloud-init`**.

Chapter 5, *Managing Storage*

Manage Ceph and Swift storage for OpenStack.

- Explain persistent storage options for use in OpenStack, focusing on the expanding capabilities of the default Ceph storage.
- Discuss the Swift and Ceph object stores, comparing architectural considerations.
- Describe the methods for configuring the new shared file systems component.
- Interpreting the behavior of ephemeral storage configuration choices.

Chapter 6, *Managing OpenStack Networking*

Manage and troubleshoot virtual network infrastructure.

- Explain the different network types available to the OpenStack networking service.
- Improve network performance with Open Virtual Networking.
- Compare the provisioning options for self-service and provider networks, and compare centralized and distributed networks.

Chapter 7, *Managing Compute Resources*

Describe the processes and architecture of both regular and hyperconverged compute nodes. Perform common administration tasks for compute nodes.

- Explain the end-to-end launch process for server instances, including scheduler and conductor tasks.
- Describe the hyperconverged compute node architecture and manage scheduling methods for sets of compute nodes.
- Perform common compute node administration tasks, including live migration, evacuation, and enabling and disabling compute nodes.

Chapter 8, *Automating Cloud Applications*

Implement complex and scalable applications using orchestration.

- Explain the orchestration architecture required to deploy application stacks.
- Write templates using the Heat Orchestration Template (HOT) language to deploy application stacks.

Chapter 9, *Troubleshooting OpenStack Operations*

Holistically diagnose and troubleshoot OpenStack issues.

- Discuss recommended diagnostic and troubleshooting tools and techniques.
- Discuss common problem scenarios for selected core components.

BUILDING A CUSTOM IMAGE

PERFORMANCE CHECKLIST

In this review, you will build a custom image and launch an instance using the custom image.

OUTCOMES

You should be able to:

- Prepare to launch an instance.
- Build an image with customizations.
- Create an SSH key pair to access instances.
- Launch an instance using the new image.
- Associate a floating IP address to the instance.
- Create security group rules to allow FTP traffic to and from the instance.

BEFORE YOU BEGIN

WARNING

Before starting the following exercises, you must reset your `workstation`, `director`, and `overcloud` virtual machines. This will destroy any data stored on those machines from previous exercises. If you want to keep any data from the lab environment, save it before resetting your machines. You must reset all machines in the lab environment for the following exercises to work as expected.

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

On `workstation`, run the **`lab image-cr setup`** command. This command ensures that the OpenStack services are running and the environment is appropriately configured for the lab.

```
[studentworkstation ~]$ lab image-cr setup
```

INSTRUCTIONS

Use the following instructions to build an image with customizations and launch an instance using the customized image.

- Confirm that the user, project, network, router, flavor, and security group required to launch the instance exist. Use the `architect1`, and `developer1` users' credential stored in the **`/home/student/architect1-rc`**, and **`/home/student/developer1-rc`** files respectively.

Lab Resources

RESOURCE TYPE	RESOURCE NAME
Project	`marketing`
User	`developer1`, `architect1`
Private network	`marketing-network1`
Router	`marketing-router1`
Flavor	`marketing-flavor`
Security group	`marketing-secgroup`

- On `workstation`, retrieve the `http://materials.example.com/osp-small.qcow2` disk image and save it under **`/home/student/`**.
- Create a copy of the **`/usr/share/diskimage-builder/elements`** directory to work with under **`/home/student/`**.
- Create a **`post-install.d`** directory under the working copy of the **`rhel7`** element.
- Create two scripts under the **`rhel7`** element of the **`post-install.d`** phase directory to enable the `vsftpd` service and configure it such that anonymous FTP users can upload files to the FTP server.
- Set the executable permission on the scripts.
- Export the required environment variables for the **`disk-image-create`** command to successfully build the custom image.
- Build the **`marketing-rhel-ftp.qcow2`** disk image and include the *vsftpd* package. The scripts created earlier are automatically integrated. Enter `student` as the password for the `student` user when prompted.
- Use the **`guestfish`** command to update the **`/etc/motd`** file of the **`marketing-rhel-ftp.qcow2`** image file to include a custom message.
- As the `developer1` OpenStack user, upload the **`marketing-rhel-ftp.qcow2`** disk image to the OpenStack image service as `marketing-rhel-ftp`, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.
- As `developer1`, create a new SSH key pair named `marketing-keypair` in the `marketing` project. Save the private key component of the key pair to **`/home/student/.ssh/marketing-keypair.pem`**.
- Associate an available floating IP address to the `marketing-ftp-server` instance.
- Add the new security group rules to the existing `marketing-secgroup` security group that allow FTP connection requests and data traffic to the `marketing-ftp-server` instance.
- Verify the FTP service and the custom message of the day configured in the instance.

Evaluation

On `workstation`, run the **`lab image-cr grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~(developer1-marketing)]$ lab image-cr grade
```

Cleanup

On `workstation`, run the **`lab image-cr cleanup`** command to clean up this exercise.

```
[student@workstation ~(developer1-marketing)]$ lab image-cr cleanup
```

This concludes the lab.

BUILDING A CUSTOM IMAGE

PERFORMANCE CHECKLIST

In this review, you will build a custom image and launch an instance using the custom image.

OUTCOMES

You should be able to:

- Prepare to launch an instance.
- Build an image with customizations.
- Create an SSH key pair to access instances.
- Launch an instance using the new image.
- Associate a floating IP address to the instance.
- Create security group rules to allow FTP traffic to and from the instance.

BEFORE YOU BEGIN

WARNING

Before starting the following exercises, you must reset your `workstation`, `director`, and `overcloud` virtual machines. This will destroy any data stored on those machines from previous exercises. If you want to keep any data from the lab environment, save it before resetting your machines. You must reset all machines in the lab environment for the following exercises to work as expected.

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

On `workstation`, run the **`lab image-cr setup`** command. This command ensures that the OpenStack services are running and the environment is appropriately configured for the lab.

```
[studentworkstation ~]$ lab image-cr setup
```

INSTRUCTIONS

Use the following instructions to build an image with customizations and launch an instance using the customized image.

- Confirm that the user, project, network, router, flavor, and security group required to launch the instance exist. Use the `architect1`, and `developer1` users' credential stored in the **`/home/student/architect1-rc`**, and **`/home/student/developer1-rc`** files respectively.

Lab Resources

RESOURCE TYPE	RESOURCE NAME
Project	`marketing`
User	`developer1`, `architect1`
Private network	`marketing-network1`
Router	`marketing-router1`
Flavor	`marketing-flavor`
Security group	`marketing-secgroup`

- On `workstation`, retrieve the `http://materials.example.com/osp-small.qcow2` disk image and save it under **`/home/student/`**.
- Create a copy of the **`/usr/share/diskimage-builder/elements`** directory to work with under **`/home/student/`**.
- Create a **`post-install.d`** directory under the working copy of the **`rhel7`** element.
- Create two scripts under the **`rhel7`** element of the **`post-install.d`** phase directory to enable the `vsftpd` service and configure it such that anonymous FTP users can upload files to the FTP server.
- Set the executable permission on the scripts.
- Export the required environment variables for the **`disk-image-create`** command to successfully build the custom image.
- Build the **`marketing-rhel-ftp.qcow2`** disk image and include the *vsftpd* package. The scripts created earlier are automatically integrated. Enter `student` as the password for the `student` user when prompted.
- Use the **`guestfish`** command to update the **`/etc/motd`** file of the **`marketing-rhel-ftp.qcow2`** image file to include a custom message.
- As the `developer1` OpenStack user, upload the **`marketing-rhel-ftp.qcow2`** disk image to the OpenStack image service as `marketing-rhel-ftp`, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.
- As `developer1`, create a new SSH key pair named `marketing-keypair` in the `marketing` project. Save the private key component of the key pair to **`/home/student/.ssh/marketing-keypair.pem`**.
- Associate an available floating IP address to the `marketing-ftp-server` instance.
- Add the new security group rules to the existing `marketing-secgroup` security group that allow FTP connection requests and data traffic to the `marketing-ftp-server` instance.
- Verify the FTP service and the custom message of the day configured in the instance.

1. Confirm that the user, project, network, router, flavor, and security group required to launch the instance exist.

1.1. As `architect1`, verify the role assignment of the `developer1` user. Use `Example` as the user domain.

```
[student@workstation ~]$ source architect1-marketing-rc
```

```
[student@workstation ~(architect1-marketing)]$ openstack role assignment list \
--name --user developer1 \
--user-domain Example
+----------+-------------------+-------+-------------------+--------+-----------+
| Role     | User              | Group | Project           | Domain | Inherited |
+----------+-------------------+-------+-------------------+--------+-----------+
| _member_ | developer1@Example |       | marketing@Default |        | False     |
+----------+-------------------+-------+-------------------+--------+-----------+
```

The preceding output shows that the user `developer1` belongs to the `marketing` project with the `_member_` role.

1.2. As `developer1`, verify that the `marketing-network` network exists.

```
[student@workstation ~(architect1-marketing)]$ source developer1-marketing-rc
[student@workstation ~(developer1-marketing)]$ openstack network list
+-------------+--------------------+-------------+
| ID          | Name               | Subnets     |
+-------------+--------------------+-------------+
| 8fda...4666 | marketing-network1 | 5dc9...c143 |
| d55f...a220 | provider-storage   | 6e5a...1d11 |
| fc54...efb3 | provider-datacentre | 30c0...33f1 |
+-------------+--------------------+-------------+
```

1.3. Verify that the `marketing-router1` router exists.

```
[student@workstation ~(developer1-marketing)]$ openstack router list -c Name
+------------------+
| Name             |
+------------------+
| marketing-router1 |
+------------------+
```

1.4. Verify that the `marketing-flavor1` flavor exists.

```
[student@workstation ~(developer1-marketing)]$ openstack flavor list -c Name
+-------------------+
| Name              |
+-------------------+
| default-extra-disk |
| default           |
| default-swap      |
| marketing-flavor  |
+-------------------+
```

1.5. Verify that the `marketing-secgroup` security group exists.

```
[student@workstation ~(developer1-marketing)]$ openstack security group list \
-c Name
```

```
+--------------------+
| Name               |
+--------------------+
| default            |
| marketing-secgroup |
+--------------------+
```

2. On `workstation`, retrieve the `http://materials.example.com/osp-small.qcow2` disk image and save it under **`/home/student/`**.

```
[student@workstation ~(developer1-marketing)]$ wget \
http://materials.example.com/osp-small.qcow2
--2018-11-22 08:40:58--  http://materials.example.com/osp-small.qcow2
Resolving materials.example.com (materials.example.com)... 172.25.254.254
Connecting to materials.example.com (materials.example.com)|172.25.254.254|:80...
 connected.
HTTP request sent, awaiting response... 200 OK
Length: 1632174080 (1.5G)
Saving to: ‘osp-small.qcow2’

100%[===========================================>] 1,632,174,080 40.5MB/s   in 28s

2018-11-22 08:41:26 (55.6 MB/s) - ‘osp-small.qcow2’ saved [1632174080/1632174080]
```

3. Create a copy of the **`/usr/share/diskimage-builder/elements`** directory to work with under **`/home/student/`**.

```
[student@workstation ~(developer1-marketing)]$ cp -a \
/usr/share/diskimage-builder/elements /home/student/
```

4. Create a **`post-install.d`** directory under the working copy of the **`rhel7`** element.

```
[student@workstation ~(developer1-marketing)]$ mkdir -p \
/home/student/elements/rhel7/post-install.d
```

5. Create two scripts under the **`rhel7`** element of the **`post-install.d`** phase directory to enable the `vsftpd` service and configure it such that anonymous FTP users can upload files to the FTP server.

   5.1. Create the first script (**`/home/student/elements/rhel7/post-install.d/01-enable-vsftpd`**) with the following content:

```
#!/bin/bash
systemctl enable vsftpd
```

   5.2. Create the second script (**`/home/student/elements/rhel7/post-install.d/02-configure-vsftpd`**) with the following content:

```
#!/bin/bash

# Enable upload access for the anonymous users
sed -i -e 's/.*anon_upload_enable=.*/anon_upload_enable=YES/' \
/etc/vsftpd/vsftpd.conf

# Create upload directory
```

```
mkdir /var/ftp/upload

# Adjust ownership and permissions
chgrp ftp /var/ftp/upload/
chmod g+w /var/ftp/upload/
```

6. Change to the **/home/student** directory and set the executable permission on the scripts.

```
[student@workstation post-install.d(developer1-marketing)]$ cd
[student@workstation ~(developer1-marketing)]$ chmod +x \
/home/student/elements/rhel7/post-install.d/*
```

7. Export the following environment variables, which diskimage-builder requires.

Environment Variables

VARIABLE	CONTENT
DIB_LOCAL_IMAGE	**/home/student/osp-small.qcow2**
DIB_YUM_REPO_CONF	**/etc/yum.repos.d/openstack.repo**
ELEMENTS_PATH	**/home/student/elements**
DIB_NO_TMPFS	**1**

```
[student@workstation ~(developer1-marketing)]$ export \
DIB_LOCAL_IMAGE=/home/student/osp-small.qcow2
[student@workstation ~(developer1-marketing)]$ export \
DIB_YUM_REPO_CONF=/etc/yum.repos.d/openstack.repo
[student@workstation ~(developer1-marketing)]$ export \
ELEMENTS_PATH=/home/student/elements
[student@workstation ~]$ export DIB_NO_TMPFS=1
```

8. Build the **marketing-rhel-ftp.qcow2** disk image and include the *vsftpd* package. The scripts created earlier are automatically integrated. Enter `student` as the password for the `student` user when prompted.

```
[student@workstation ~(developer1-marketing)]$ disk-image-create vm rhel7 \
-t qcow2 \
-p vsftpd \
-o marketing-rhel-ftp.qcow2
...output omitted...
```

9. Use the **guestfish** command to update the **/etc/motd** file of the **marketing-rhel-ftp.qcow2** image file to include a custom message.

```
[student@workstation ~(developer1-marketing)]$ guestfish -i -a \
~/marketing-rhel-ftp.qcow2

Welcome to guestfish, the guest filesystem shell for
editing virtual machine filesystems and disk images.

Type: 'help' for help on commands
```

```
      'man' to read the manual
      'quit' to quit the shell

Operating system: Red Hat Enterprise Linux Server 7.5 (Maipo)
/dev/sda1 mounted on /

><fs> write /etc/motd "Hello World"
><fs> cat /etc/motd
Hello World
><fs> selinux-relabel /etc/selinux/targeted/contexts/files/file_contexts /
><fs> exit
```

IMPORTANT
The preceding **selinux-relabel** command applies the appropriate SELinux file contexts to the root file system recursively.

10. As the developer1 OpenStack user, upload the **marketing-rhel-ftp.qcow2** disk image to the OpenStack Image service as marketing-rhel-ftp, with a minimum disk requirement of 10 GiB, and a minimum RAM requirement of 2 GiB.

10.1. As developer1, use the **openstack** command to upload the disk image **marketing-rhel-ftp.qcow2** into the OpenStack Image service.

```
[student@workstation ~(developer1-marketing)]$ openstack image create \
--disk-format qcow2 \
--min-disk 10 \
--min-ram 2048 \
--file ~/marketing-rhel-ftp.qcow2 \
marketing-rhel-ftp
...output omitted...
```

11. As developer1, create a new SSH key pair named marketing-keypair in the marketing project. Save the private key component of the key pair to **/home/student/.ssh/marketing-keypair.pem**. Set **600** as the file permissions for **/home/student/.ssh/marketing-keypair.pem**.

```
[student@workstation ~(developer1-marketing)]$ openstack keypair create \
marketing-keypair > /home/student/.ssh/marketing-keypair.pem
[student@workstation ~(developer1-marketing)]$ chmod 600 \
~/.ssh/marketing-keypair.pem
```

12. Launch an instance in the environment using the following attributes. Use the user-data file **/home/student/cr1-user-data** to customize the SELinux settings of the marketing-ftp-server instance.

Instance Attributes

ATTRIBUTE	VALUE
flavor	**marketing-flavor**
key pair	**marketing-keypair**
network	**marketing-network1**

ATTRIBUTE	VALUE
image	marketing-rhel-ftp
security group	marketing-secgroup
user data file	/home/student/cr1-user-data
name	marketing-ftp-server

```
[student@workstation ~(developer1-marketing)]$ openstack server create \
--flavor marketing-flavor \
--key-name marketing-keypair \
--nic net-id=marketing-network1 \
--image marketing-rhel-ftp \
--security-group marketing-secgroup \
--user-data cr1-user-data \
--wait marketing-ftp-server
...output omitted...
```

13. List the available floating IP addresses, and then allocate one to `marketing-ftp-server`.

13.1. List the available floating IP addresses. The floating IP addresses available on your system may differ from those shown below.

```
[student@workstation ~(developer1-marketing)]$ openstack floating ip list \
-c "Floating IP Address" -c Port
+---------------------+------+
| Floating IP Address | Port |
+---------------------+------+
| 172.25.250.101      | None |
+---------------------+------+
```

13.2. Associate an available floating IP address to the `marketing-ftp-server` instance.

```
[student@workstation ~(developer1-marketing)]$ openstack server add floating \
ip marketing-ftp-server 172.25.250.101
```

14. Add the new security group rules to the existing `marketing-secgroup` security group that allow FTP connection requests and data traffic to the `marketing-ftp-server` instance.

```
[student@workstation ~(developer1-marketing)]$ openstack security group \
rule create \
--dst-port 21:21 \
--protocol tcp \
marketing-secgroup
...output omitted...
[student@workstation ~(developer1-marketing)]$ openstack security group \
rule create \
--dst-port 20:20 \
--protocol tcp \
marketing-secgroup
...output omitted...
```

15. If the image build and customization were successful, the instance created using this image will have the FTP service activated for anonymous FTP users to upload files.

15.1. Use the **`ftp`** command to connect to the FTP service in the `marketing-ftp-server` instance and upload the **`/home/student/developer1-marketing-rc`** file to the FTP server as the `ftp` anonymous user.

```
[student@workstation ~(developer1-marketing)]$ ftp -A 172.25.250.101
Connected to 172.25.250.101 (172.25.250.101).
220 (vsFTPd 3.0.2)
Name (172.25.250.101:student): ftp
331 Please specify the password.
Password: Enter
230 Login successful.
Remote system type is UNIX.
Using binary mode to transfer files.
ftp> cd upload
250 Directory successfully changed.
ftp> pwd
257 "/upload"
ftp> put developer1-marketing-rc
local: developer1-marketing-rc remote: developer1-marketing-rc
200 PORT command successful. Consider using PASV.
150 Ok to send data.
226 Transfer complete.
347 bytes sent in 3.9e-05 secs (8897.44 Kbytes/sec)
ftp> ls
200 PORT command successful. Consider using PASV.
150 Here comes the directory listing.
-rw-------    1 14       50            347 Nov 22 12:34 developer1-marketing-rc
226 Directory send OK.
ftp> exit
221 Goodbye.
```

The **`-A`** option in the preceding **`ftp`** command causes the FTP client to run in active mode rather than passive mode.

15.2. Verify the custom message set in the **`/etc/motd`** file of the instance.

```
[student@workstation ~(developer1-marketing)]$ ssh -i \
~/.ssh/marketing-keypair.pem cloud-user@172.25.250.101
Last login: Thu Nov 22 12:12:57 2018 from workstation.lab.example.com
Hello World
[cloud-user@marketing-ftp-server ~]$
```

Notice the custom message in the preceding output that appears right after the successful SSH authentication.

15.3. Log off from the `marketing-ftp-server` instance.

```
[cloud-user@marketing-ftp-server ~]$ logout
```

Evaluation

On `workstation`, run the **`lab image-cr grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~(developer1-marketing)]$ lab image-cr grade
```

Cleanup

On `workstation`, run the **`lab image-cr cleanup`** command to clean up this exercise.

```
[student@workstation ~(developer1-marketing)]$ lab image-cr cleanup
```

This concludes the lab.

MANAGING PROJECT ORGANIZATION

PERFORMANCE CHECKLIST

In this review, you will create nested projects within a domain and assign roles in OpenStack to users authenticated by the Identity Service.

OUTCOMES

You should be able to:

- Create domains, projects, and subprojects.
- Create users and assign user roles.
- Create groups and assign group roles.
- Identify roles assigned to users in a domain.

BEFORE YOU BEGIN

WARNING

Before starting the following exercises, you must reset your `workstation`, `director`, and `overcloud` virtual machines. This will destroy any data stored on those machines from previous exercises. If you want to keep any data from the lab environment, save it before resetting your machines. You must reset all machines in the lab environment for the following exercises to work as expected.

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

On `workstation`, run the **`lab identity-cr setup`** command. This command will ensure that the OpenStack services are running and the environment is properly configured for the lab.

```
[studentworkstation ~]$ lab identity-cr setup
```

INSTRUCTIONS

Use the following instructions to set up the nested projects within a domain and assign roles in OpenStack to users authenticated by the Identity Service.

- Create a new domain named `Lab`
- Create a project named `marketing` in the `Lab` domain.
- Create a subproject named `development` in the `marketing` project.
- Create a new user group named `developers` and `admins` within the `Lab` domain.
- Assign the `developers` group in the `Lab` domain the `_member_` role for the `development` project.

- Assign the `admins` group in the `Lab` domain the `admin` role for the `development` project.
- Create two users named `operator1`, and `architect1` in the `Lab` domain. Set the password for both users to `redhat`.
- Add the `operator1` user to the `developers` user group of the `Lab` domain.
- Add the `architect1` user to the `admins` user group of the `Lab` domain.
- Assign the `operator1` user in the `Lab` domain the `_member_` role for the `marketing` project in the `Lab` domain.
- Assign the `architect1` user in the `Lab` domain the `admin` role for the `marketing` project in the `Lab` domain.
- Verify the role assigned to the `operator1` and `architect1` users in the different projects.

Evaluation

On `workstation`, run the **`lab identity-cr grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab identity-cr grade
```

Cleanup

On `workstation`, run the **`lab identity-cr cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab identity-cr cleanup
```

This concludes the lab.

MANAGING PROJECT ORGANIZATION

PERFORMANCE CHECKLIST

In this review, you will create nested projects within a domain and assign roles in OpenStack to users authenticated by the Identity Service.

OUTCOMES

You should be able to:

- Create domains, projects, and subprojects.
- Create users and assign user roles.
- Create groups and assign group roles.
- Identify roles assigned to users in a domain.

BEFORE YOU BEGIN

WARNING

Before starting the following exercises, you must reset your `workstation`, `director`, and `overcloud` virtual machines. This will destroy any data stored on those machines from previous exercises. If you want to keep any data from the lab environment, save it before resetting your machines. You must reset all machines in the lab environment for the following exercises to work as expected.

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

On `workstation`, run the **`lab identity-cr setup`** command. This command will ensure that the OpenStack services are running and the environment is properly configured for the lab.

```
[studentworkstation ~]$ lab identity-cr setup
```

INSTRUCTIONS

Use the following instructions to set up the nested projects within a domain and assign roles in OpenStack to users authenticated by the Identity Service.

- Create a new domain named `Lab`
- Create a project named `marketing` in the `Lab` domain.
- Create a subproject named `development` in the `marketing` project.
- Create a new user group named `developers` and `admins` within the `Lab` domain.
- Assign the `developers` group in the `Lab` domain the `_member_` role for the `development` project.
- Assign the `admins` group in the `Lab` domain the `admin` role for the `development` project.

- Create two users named `operator1`, and `architect1` in the `Lab` domain. Set the password for both users to `redhat`.
- Add the `operator1` user to the `developers` user group of the `Lab` domain.
- Add the `architect1` user to the `admins` user group of the `Lab` domain.
- Assign the `operator1` user in the `Lab` domain the `_member_` role for the `marketing` project in the `Lab` domain.
- Assign the `architect1` user in the `Lab` domain the `admin` role for the `marketing` project in the `Lab` domain.
- Verify the role assigned to the `operator1` and `architect1` users in the different projects.

1. On `workstation`, source the `admin` environment file, **`/home/student/admin-rc`**. Create a domain named `Lab`.

```
[student@workstation ~]$ source ~/admin-rc
[student@workstation ~]$ openstack domain create Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| enabled     | True                             |
| id          | f4b6864041e945dfa55af34592919c8a |
| name        | Lab                              |
| tags        | []                               |
+-------------+----------------------------------+
```

2. Create a project named `marketing` and a subproject named `development` within the `Lab` domain.

   2.1. Create a project named `marketing` within the `Lab` domain.

```
[student@workstation ~]$ openstack project create marketing --domain Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | f4b6864041e945dfa55af34592919c8a |
| enabled     | True                             |
| id          | 5e2af53627d24004a218989be21262de |
| is_domain   | False                            |
| name        | marketing                        |
| parent_id   | f4b6864041e945dfa55af34592919c8a |
| tags        | []                               |
+-------------+----------------------------------+
```

   2.2. Create a subproject named `development` with the `marketing` project as the parent within the `Lab` domain.

```
[student@workstation ~]$ openstack project create \
development --parent marketing --domain Lab
```

```
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | f4b6864041e945dfa55af34592919c8a |
| enabled     | True                             |
| id          | dbdbfec36b164c6ab64376c2ae4c3c22 |
| is_domain   | False                            |
| name        | development                      |
| parent_id   | 5e2af53627d24004a218989be21262de |
| tags        | []                               |
+-------------+----------------------------------+
```

3. Create user groups named `developers` and `admins` within the `Lab` domain.

3.1. Create a user group named `developers` within the `Lab` domain.

```
[student@workstation ~]$ openstack group create developers --domain Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | f4b6864041e945dfa55af34592919c8a |
| id          | 6dfb18c4553144eab0efb7ca268c2432 |
| name        | developers                       |
+-------------+----------------------------------+
```

3.2. Create a user group named `admins` within the `Lab` domain.

```
[student@workstation ~]$ openstack group create admins --domain Lab
+-------------+----------------------------------+
| Field       | Value                            |
+-------------+----------------------------------+
| description |                                  |
| domain_id   | f4b6864041e945dfa55af34592919c8a |
| id          | e856ac04b487473c8af240eb138cc124 |
| name        | admins                           |
+-------------+----------------------------------+
```

4. Assign the `developers` group in the `Lab` domain the `_member_` role for the `development` project in the `Lab` domain.

```
[student@workstation ~]$ openstack role add \
--project development --project-domain Lab \
--group developers --group-domain Lab \
_member_
```

5. Assign the `admins` group in the `Lab` domain the `admin` role for the `development` project in the `Lab` domain.

```
[student@workstation ~]$ openstack role add \
--project development --project-domain Lab \
--group admins --group-domain Lab \
admin
```

6. Create two users named `operator1` and `architect1` in the `Lab` domain. Set the password for both users to `redhat`.

   6.1. Create a user named `operator1` in the `Lab` domain. Set the password for the user to `redhat`.

```
[student@workstation ~]$ openstack user create \
--domain Lab --password redhat operator1
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | f4b6864041e945dfa55af34592919c8a |
| enabled             | True                             |
| id                  | 88b7672f6a1e4f47a843a6d39f2aec19 |
| name                | operator1                        |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

   6.2. Create a user named `architect1` in the `Lab` domain. Set the password for the user to `redhat`.

```
[student@workstation ~]$ openstack user create \
--domain Lab --password redhat architect1
+---------------------+----------------------------------+
| Field               | Value                            |
+---------------------+----------------------------------+
| domain_id           | f4b6864041e945dfa55af34592919c8a |
| enabled             | True                             |
| id                  | 557e0d11f6d14a4795bf9f781d09cfdb |
| name                | architect1                       |
| options             | {}                               |
| password_expires_at | None                             |
+---------------------+----------------------------------+
```

7. Add the `operator1` user to the `developers` user group, and the `architect1` user to the `admins` user group in the `Lab` domain.

   7.1. Add the `operator1` user of the `Lab` domain to the `developers` user group of the `Lab` domain.

```
[student@workstation ~]$ openstack group add user \
--group-domain Lab --user-domain Lab \
developers operator1
```

   7.2. Add the `architect1` user of the `Lab` domain to the `admins` user group of the `Lab` domain.

```
[student@workstation ~]$ openstack group add user \
--group-domain Lab --user-domain Lab \
admins architect1
```

8. Assign the `_member_` role for the `marketing` project to the `operator1` user.

```
[student@workstation ~]$ openstack role add \
--project marketing --project-domain Lab \
```

```
--user operator1 --user-domain Lab \
_member_
```

9. Assign the `admin` role for the `marketing` project to the `architect1` user.

```
[student@workstation ~]$ openstack role add \
--project marketing --project-domain Lab \
--user architect1 --user-domain Lab \
admin
```

10. Verify the role assignment to the `operator1` user in the `marketing` and `development` projects.

    10.1. Verify the role assignment for the `operator1` user in the `marketing` project.

```
[student@workstation ~]$ openstack role assignment list \
--project marketing --project-domain Lab \
--user operator1 --user-domain Lab \
--names --effective
+----------+---------------+-------+---------------+--------+-----------+
| Role     | User          | Group | Project       | Domain | Inherited |
+----------+---------------+-------+---------------+--------+-----------+
| _member_ | operator1@Lab |       | marketing@Lab |        | False     |
+----------+---------------+-------+---------------+--------+-----------+
```

    10.2. Verify the role assignment for the `operator1` user in the `development` project.

```
[student@workstation ~]$ openstack role assignment list \
--project development --project-domain Lab \
--user operator1 --user-domain Lab \
--names --effective
+----------+---------------+-------+-----------------+--------+-----------+
| Role     | User          | Group | Project         | Domain | Inherited |
+----------+---------------+-------+-----------------+--------+-----------+
| _member_ | operator1@Lab |       | development@Lab |        | False     |
+----------+---------------+-------+-----------------+--------+-----------+
```

11. Log in to the dashboard to verify the role assigned to the `architect1` user.

    11.1. On `workstation`, launch the Firefox web browser and connect to the dashboard at `http://dashboard.overcloud.example.com/`.

    11.2. Enter **Lab** for the Domain, **`architect1`** for the User Name, and **`redhat`** for the Password to authenticate.

    11.3. After you have logged in to the dashboard, observe that the top navigation menu displays the Project, Admin, and Identity tabs.

    11.4. Choose development from the Project menu to switch from the `marketing` project to the `development` project. Observe that the top navigation menu displays the Project, Admin, and Identity tabs.

    11.5. Log out of the dashboard.

Evaluation

On `workstation`, run the **`lab identity-cr grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab identity-cr grade
```

Cleanup

On `workstation`, run the **`lab identity-cr cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab identity-cr cleanup
```

This concludes the lab.

LAUNCHING AN APPLICATION STACK

PERFORMANCE CHECKLIST

In this exercise, you will write a template that will launch a stack.

OUTCOMES

You should be able to:

- Modify a simple orchestration template.
- Launch a multi-instance application stack using the orchestration template.
- Add a block storage volume.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab stack-cr setup`** command. The script downloads the orchestration templates and environment file to the **`/home/student/heat-templates`** directory.

```
[studentworkstation ~]$ lab stack-cr setup
```

INSTRUCTIONS

Use the following instructions to build an image with customizations and launch an instance using the customized image.

- Edit the **`/home/student/.vimrc`** file with the following content to set up automatic indentation.
- Edit the orchestration template **`/home/student/heat-templates/marketing-app1.yaml`** to launch an application stack comprised of one web server instance and one database server instance. The database server instance should use a persistent volume as the back-end storage for the database. The persistent volume should be provisioned as a member of the stack. The web server and the database server should use separate user-data files such as **`/home/student/heat-templates/webserver-cr3.sh`** and **`/home/student/heat-templates/dbserver-cr3.sh`** for their respective customizations. The orchestration stack must retry once to execute the user-data script. The user-data script must return success on the successful execution of the script. Set the timeout period to 600 seconds, after which the **`webserver-cr3.sh`** script must return a **`fail`** result. Set the timeout period to 800 seconds, after which the **`dbserver-cr3.sh`** script must return a **`fail`** result.
- Create the **`/home/student/heat-templates/environment-cr3.yaml`** environment file. Enter the values for all input parameters defined in the **`/home/student/heat-templates/marketing-app1.yaml`** template file.

Input Parameters

RESOURCE TYPE	RESOURCE NAME
Web server image	`rhel7-web`
Database server image	`rhel7-db`
Web server name	`marketing-web-server1`
Database server name	`marketing-db-server2`
Flavor	`default`
Key pair name	`marketing-keypair`
Public network	`provider-datacentre`
Private network	`marketing-network1`
Private subnet	`marketing-subnet1`

- Launch the stack and verify it by accessing the web page deployed on the instance. Use the `developer1` user credentials to launch the stack.

Evaluation

On `workstation`, run the **`lab stack-cr grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab stack-cr grade
```

Cleanup

On `workstation`, run the **`lab stack-cr cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab stack-cr cleanup
```

This concludes the lab.

LAUNCHING AN APPLICATION STACK

PERFORMANCE CHECKLIST

In this exercise, you will write a template that will launch a stack.

OUTCOMES

You should be able to:

- Modify a simple orchestration template.
- Launch a multi-instance application stack using the orchestration template.
- Add a block storage volume.

BEFORE YOU BEGIN

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

On `workstation`, run the **`lab stack-cr setup`** command. The script downloads the orchestration templates and environment file to the **`/home/student/heat-templates`** directory.

```
[studentworkstation ~]$ lab stack-cr setup
```

INSTRUCTIONS

Use the following instructions to build an image with customizations and launch an instance using the customized image.

- Edit the **`/home/student/.vimrc`** file with the following content to set up automatic indentation.
- Edit the orchestration template **`/home/student/heat-templates/marketing-app1.yaml`** to launch an application stack comprised of one web server instance and one database server instance. The database server instance should use a persistent volume as the back-end storage for the database. The persistent volume should be provisioned as a member of the stack. The web server and the database server should use separate user-data files such as **`/home/student/heat-templates/webserver-cr3.sh`** and **`/home/student/heat-templates/dbserver-cr3.sh`** for their respective customizations. The orchestration stack must retry once to execute the user-data script. The user-data script must return success on the successful execution of the script. Set the timeout period to 600 seconds, after which the **`webserver-cr3.sh`** script must return a **`fail`** result. Set the timeout period to 800 seconds, after which the **`dbserver-cr3.sh`** script must return a **`fail`** result.
- Create the **`/home/student/heat-templates/environment-cr3.yaml`** environment file. Enter the values for all input parameters defined in the **`/home/student/heat-templates/marketing-app1.yaml`** template file.

Input Parameters

RESOURCE TYPE	RESOURCE NAME
Web server image	`rhel7-web`
Database server image	`rhel7-db`
Web server name	`marketing-web-server1`
Database server name	`marketing-db-server2`
Flavor	`default`
Key pair name	`marketing-keypair`
Public network	`provider-datacentre`
Private network	`marketing-network1`
Private subnet	`marketing-subnet1`

- Launch the stack and verify it by accessing the web page deployed on the instance. Use the `developer1` user credentials to launch the stack.

1. Edit the **`/home/student/.vimrc`** file with the following content to set up automatic indentation:

```
autocmd FileType yaml setlocal ai ts=2 sw=2 et
```

2. Edit the orchestration template **`/home/student/heat-templates/marketing-app1.yaml`** to launch an application stack comprised of one web server instance and one database server instance. The database server instance should use a persistent volume as the back-end storage for the database. The persistent volume should be provisioned as a member of the stack.

   The orchestration template must include the following items:

   - The `web_server` resource definition deploys the instance with a PHP web page that interacts with the MariaDB database. Run the **`/home/student/heat-templates/webserver-cr3.sh`** script as the user-data script to deploy the intended PHP web page on the `marketing-web-server1` instance.

   - The `db_server` resource definition deploys the instance with a MariaDB database. Run the **`/home/student/heat-templates/dbserver-cr3.sh`** script as the user-data script to deploy the intended MariaDB database on `marketing-db-server2`. The script also creates a database named `mydb`.

   - The following parameters must be passed to the **`webserver-cr3.sh`** script using the **`user_data`** property:

     The `$web_public_ip` and `$web_private_ip` variables must be initialized with the external and private IP addresses of the `finance-web` instance, respectively. The `$db_private_ip` variable must be initialized with the private IP address of the `finance-db` instance. These variables are used in the **`webserver-cr3.sh`** script.

- The orchestration stack must retry once to execute the user-data script. The user-data script must return success on the successful execution of the script. Set the timeout period to 600 seconds, after which the **`webserver-cr3.sh`** script must return a **`fail`** result. Set the timeout period to 800 seconds, after which the **`dbserver-cr3.sh`** script must return a **`fail`** result.

2.1. Navigate to the **`/home/student/heat-templates/`** directory.

```
[studentworkstation ~]$ cd heat-templates
[studentworkstation heat-templates]$
```

2.2. Edit the **`/home/student/heat-templates/marketing-app1.yaml`** file to specify the **`user_data`** property and set the **`/home/student/heat-templates/webserver-cr3.sh`** as the user-data script to deploy a PHP web page to `marketing-web`. Set the **`user_data_format`** property of the **`OS::Nova::Server`** resource type to **`RAW`**.

The edited **`/home/student/heat-templates/marketing-app1.yaml`** file should have the content highlighted below:

```
  web_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: web_instance_name }
      image: { get_param: web_image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: web_net_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: /home/student/heat-templates/webserver-cr3.sh }
```

2.3. The web page uses the `$web_private_ip`, `$db_private_ip`, and `$web_public_ip` variables passed as parameters. These parameters are defined using the **`params`** property of the **`str_replace`** intrinsic function.

The `$web_private_ip` variable uses the `web_net_port` resource attribute, **`fixed_ips`**, to retrieve the first IP address associated with the `web_server` resource. The `$db_private_ip` variable uses the `db_net_port` resource attribute, **`fixed_ips`**, to retrieve the first IP address associated with the `db_server` resource. The `$web_public_ip` variable uses the `web_floating_ip` resource attribute, **`floating_ip_address`**, to set the public IP address associated with the `web_server` resource.

Edit the **`/home/student/heat-templates/marketing-app1.yaml`** file and add the content highlighted below:

```
  web_server:
    type: OS::Nova::Server
    properties:
      name: { get_param: instance_name }
      image: { get_param: image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
```

```
      - port: { get_resource: web_net_port }
    user_data_format: RAW
    user_data:
      str_replace:
        template: { get_file: /home/student/heat-templates/webserver-cr3.sh }
        params:
          $web_private_ip: {get_attr: [web_net_port,fixed_ips,0,ip_address]}
          $db_private_ip: {get_attr: [db_net_port,fixed_ips,0,ip_address]}
          $web_public_ip: {get_attr: [web_floating_ip,floating_ip_address]}
```

2.4. Use the **`WaitHandleCondition`** resource to send a signal about the status of the user-data script.

The `$wc_notify` variable is set to the wait handle URL using the **`curl_cli`** attribute of the `web_wait_handle` resource. The wait handle URL value is set to the `$wc_notify` variable. If the web page deployed by the script is accessible (HTTP status code 200), then the `$wc_notify` variable returns a status of **SUCCESS**. The `web_server` resource state is marked as **CREATE_COMPLETE** when the **`WaitConditionHandle`** resource signals **SUCCESS**.

The **`WaitConditionHandle`** returns **FAILURE** if the web page is not accessible or if it times out after 600 seconds. The `web_server` resource state is then marked as **CREATE_FAILED**.

Edit the **`/home/student/heat-templates/marketing-app1.yaml`** file and add the content highlighted below:

```
web_server:
  type: OS::Nova::Server
  properties:
    name: { get_param: instance_name }
    image: { get_param: image_name }
    flavor: { get_param: instance_flavor }
    key_name: { get_param: key_name }
    networks:
      - port: { get_resource: web_net_port }
    user_data_format: RAW
    user_data:
      str_replace:
        template: { get_file: /home/student/heat-templates/webserver-cr3.sh }
        params:
          $web_private_ip: {get_attr: [web_net_port,fixed_ips,0,ip_address]}
          $db_private_ip: {get_attr: [db_net_port,fixed_ips,0,ip_address]}
          $web_public_ip: {get_attr: [web_floating_ip,ip,floating_ip_address]}
          $wc_notify: {get_attr: [web_wait_handle,curl_cli]}
```

2.5. Use the **`user_data`** property to define the **`/home/student/heat-templates/dbserver-cr3.sh`** user-data script to configure the MariaDB server on `finance-db`. Set the **`user_data_format`** property of the **`OS::Nova::Server`** resource type to **RAW**.

Use the **`WaitHandleCondition`** resource to send a signal about the status of the user-data script. The `$wc_notify` variable is set to the wait handle URL using the **`curl_cli`** attribute of the `db_wait_handle` resource.

Edit the **`/home/student/heat-templates/marketing-app1.yaml`** file and add the content highlighted below:

```
db_server:
```

```
    type: OS::Nova::Server
    properties:
      name: { get_param: db_instance_name }
      image: { get_param: db_image_name }
      flavor: { get_param: instance_flavor }
      key_name: { get_param: key_name }
      networks:
        - port: { get_resource: web_net_port }
      user_data_format: RAW
      user_data:
        str_replace:
          template: { get_file: /home/student/heat-templates/dbserver-cr3.sh }
          params:
            $db_wc_notify: {get_attr: [db_wait_handle,curl_cli]}
```

2.6. Add a new resource of the `OS::Cinder::Volume` type to create a persistent volume for the `marketing-db-server2` instance to use as the database volume. Create the parameters that define the availability zone, size, and the mount point for the volume. Edit the **`/home/student/heat-templates/marketing-app1.yaml`** file to include the following definitions for the volume resource and the relevant parameters.

```
parameters:
...output omitted...
  availability_zone:
    type: string
    description: The Availability Zone to launch the instance.
    default: nova
  volume_size:
    type: number
    description: Size of the volume to be created.
    default: 1
  volume_mountpoint:
    type: string
    description: The location where the volume is exposed on the instance.
    default: /dev/vdb

resources:
...output omitted...
  cinder_volume:
    type: OS::Cinder::Volume
    properties:
      size: { get_param: volume_size }
      availability_zone: { get_param: availability_zone }
  cinder_volume_attachment:
    type: OS::Cinder::VolumeAttachment
    properties:
      volume_id: { get_resource: cinder_volume }
      instance_uuid: { get_resource: db_server }
      mountpoint: { get_param: volume_mountpoint }
```

NOTE

You may also download the ready-to-use version of the **`marketing-app1.yaml`** file from `http://materials.example.com/heat/marketing-app1-final.yaml`.

3. Create the **/home/student/heat-templates/environment-cr3.yaml** environment file. Enter the values for all input parameters defined in the **/home/student/heat-templates/marketing-app1.yaml** template file.

   Create the **/home/student/heat-templates/environment-cr3.yaml** file with the following content:

```
parameters:
  web_image_name: rhel7-web
  db_image_name: rhel7-db
  web_instance_name: marketing-web-server1
  db_instance_name: marketing-db-server2
  instance_flavor: default
  key_name: marketing-keypair
  public_net: provider-datacentre
  private_net: marketing-network1
  private_subnet: marketing-subnet1
```

4. Launch the stack and verify it by accessing the web page deployed on the instance. Use the `developer1` user credentials to launch the stack.

   4.1. Use the `developer1` user credentials to perform a dry-run stack deployment and inspect the resources that will be created when you launch the stack. Rectify all errors before proceeding to the next step.

   Use the **marketing-app1.yaml** template file and the **environment-cr3.yaml** environment file to create the stack. Name the stack `marketing-app1`.

```
[student@workstation heat-templates]$ source ~/developer1-marketing-rc
[student@workstation heat-templates(developer1-marketing)]$ openstack stack \
create \
--environment /home/student/heat-templates/environment-cr3.yaml \
--template /home/student/heat-templates/marketing-app1.yaml \
--dry-run -c description \
marketing-app1
+-------------+------------------------------+
| Field       | Value                        |
+-------------+------------------------------+
| description | multi-instance stack template |
+-------------+------------------------------+
```

   4.2. Launch the stack using the **marketing-app1.yaml** template file and the **environment-cr3.yaml** environment file. Name the stack `marketing-app1`.

   If the dry run is successful, use the **openstack stack create** command to launch the stack. Do not use the **--dry-run** option while launching the stack.

```
[student@workstation heat-templates(developer1-marketing)]$ openstack stack \
create \
--environment /home/student/heat-templates/environment-cr3.yaml \
--template /home/student/heat-templates/marketing-app1.yaml \
--wait \
marketing-app1
```

```
...output omitted...
```

4.3. List the output returned by the `marketing-app1` stack.

```
[student@workstation heat-templates(developer1-marketing)]$ openstack stack \
output list marketing-app1
+----------------+--------------------------------------------------+
| output_key     | description                                      |
+----------------+--------------------------------------------------+
| web_private_ip | Private IP address of the web server             |
| web_public_ip  | External IP address of the web server            |
| website_url    | This URL is the "external" URL that can be used  |
|                | to access the web server.                        |
|                |                                                  |
| db_private_ip  | Private IP address of the DB server              |
+----------------+--------------------------------------------------+
```

4.4. Inspect the **`website_url`** output value of the stack. The IP address in the **`output_value`** field might vary.

```
[student@workstation heat-templates(developer1-marketing)]$ openstack stack \
output show marketing-app1 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://172.25.250.105/index.php           |
+--------------+-------------------------------------------+
```

4.5. Verify that the multitier stack was created and the user-data script was executed successfully on both the instances by accessing the URL returned as the value for the **`website_url`** output.

5. Access the `marketing-db-server2` instance and configure the `mariadb` service to use the persistent block storage volume (`vdb`) of 1 GB capacity as its data directory.

```
[student@workstation heat-templates(developer1-marketing)]$ openstack server \
list -c Name -c Networks
+-----------------------+-------------------------------------------------+
| Name                  | Networks                                        |
+-----------------------+-------------------------------------------------+
| marketing-web-server1 | marketing-network1=192.168.1.10, 172.25.250.105 |
| marketing-db-server2  | marketing-network1=192.168.1.13, 172.25.250.102 |
+-----------------------+-------------------------------------------------+
[student@workstation heat-templates(developer1-marketing)]$ ssh \
-i ~/.ssh/marketing-keypair cloud-user@172.25.250.102
[cloud-user@marketing-db-server2 ~]$ lsblk -fs
NAME  FSTYPE LABEL UUID                                 MOUNTPOINT
vda1  xfs          6bea2b7b-e6cc-4dba-ac79-be6530d348f5 /
└─vda
vdb   xfs          488eb97f-83d2-4067-8a4c-c79d35970044
[cloud-user@marketing-db-server2 ~]$ sudo mount \
/dev/vdb /var/lib/mysql
```

```
[cloud-user@marketing-db-server2 ~]$ sudo chcon \
-Rt mysqld_db_t /var/lib/mysql/
[cloud-user@marketing-db-server2 ~]$ sudo chown \
-R mysql:mysql /var/lib/mysql/
[cloud-user@marketing-db-server2 ~]$ sudo systemctl \
restart mariadb.service
```

The persistent block storage volume was added as a member of the `marketing-app1` stack. The `mysqld_db_t` SELinux context allows the database service access the **`/var/lib/mysql`** data directory. Because the database service runs as the `mysql` user, the `mysql` user and `mysql` group should own the **`/var/lib/mysql`** data directory.

6. On `marketing-db-server2`, download the `http://materials.example.com/heat/scripts/test_db.dump` to **`/home/cloud-user/`**. Restore the **`/home/cloud-user/test_db.dump`** file to the `test` database in the `marketing-db-server2` MariaDB server and verify that the new database table `tbl_1` exists with an entry.

```
[cloud-user@marketing-db-server2 ~]$ curl -s -o test_db.dump \
http://materials.example.com/heat/scripts/test_db.dump
[cloud-user@marketing-db-server2 ~]$ mysql -u root test < test_db.dump
[cloud-user@marketing-db-server2 ~]$ mysql -u root test -e "select * from tbl_1;"
+------+------+
| id   | name |
+------+------+
| 1    | One  |
+------+------+
```

7. Log off from `marketing-db-server2`.

```
[cloud-user@marketing-db-server2 ~]$ logout
[student@workstation heat-templates(developer1-marketing)]$
```

Evaluation

On `workstation`, run the **`lab stack-cr grade`** command to confirm success of this exercise. Correct any reported failures and rerun the command until successful.

```
[student@workstation ~]$ lab stack-cr grade
```

Cleanup

On `workstation`, run the **`lab stack-cr cleanup`** command to clean up this exercise.

```
[student@workstation ~]$ lab stack-cr cleanup
```

This concludes the lab.