Red Hat OpenStack Platform 17.1

Configuring load balancing as a service

Managing network traffic across the data plane using the Load-balancing service (octavia)

Last Updated: 2024-05-23

Red Hat OpenStack Platform 17.1 Configuring load balancing as a service

Managing network traffic across the data plane using the Load-balancing service (octavia)

OpenStack Team
rhos-docs@redhat.com

Legal Notice

Abstract

Install, configure, operate, troubleshoot, and upgrade the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia).

Table of Contents

MAKING OPEN SOURCE MORE INCLUSIVE

Red Hat is committed to replacing problematic language in our code, documentation, and web properties. We are beginning with these four terms: master, slave, blacklist, and whitelist. Because of the enormity of this endeavor, these changes will be implemented gradually over several upcoming releases. For more details, see our CTO Chris Wright’s message .

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Tell us how we can make it better.

Providing documentation feedback in Jira

Use the Create Issue form to provide feedback on the documentation. The Jira issue will be created in the Red Hat OpenStack Platform Jira project, where you can track the progress of your feedback.

1. Ensure that you are logged in to Jira. If you do not have a Jira account, create an account to submit feedback.

2. Click the following link to open a the **Create Issue** page: Create Issue

3. Complete the **Summary** and **Description** fields. In the **Description** field, include the documentation URL, chapter or section number, and a detailed description of the issue. Do not modify any other fields in the form.

4. Click **Create**.

CHAPTER 1. INTRODUCTION TO THE LOAD-BALANCING SERVICE

The Load-balancing service (octavia) provides a Load Balancing-as-a-Service (LBaaS) API version 2 implementation for Red Hat OpenStack Platform (RHOSP) deployments. The Load-balancing service manages multiple virtual machines, containers, or bare metal servers—collectively known as amphorae—which it launches on demand. The ability to provide on-demand, horizontal scaling makes the Load-balancing service a fully-featured load balancer that is appropriate for large RHOSP enterprise deployments.

NOTE

Red Hat does not support a migration path from Neutron-LBaaS to the Load-balancing service. You can use some unsupported open source tools. For example, search nlbaas2octavia-lb-replicator on GitHub.

- Section 1.1, “Load-balancing service components”
- Section 1.2, “Load-balancing service object model”
- Section 1.3, “Uses of load balancing in Red Hat OpenStack Platform”

1.1. LOAD-BALANCING SERVICE COMPONENTS

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) uses a set of VM instances referred to as *amphorae* that reside on the Compute nodes. The Load-balancing service controllers communicate with the amphorae over a load-balancing management network (**lb-mgmt-net**).

When using octavia, you can create load-balancer virtual IPs (VIPs) that do not require floating IPs (FIPs). Not using FIPs has the advantage of improving performance through the load balancer.

Figure 1.1. Load-balancing service components

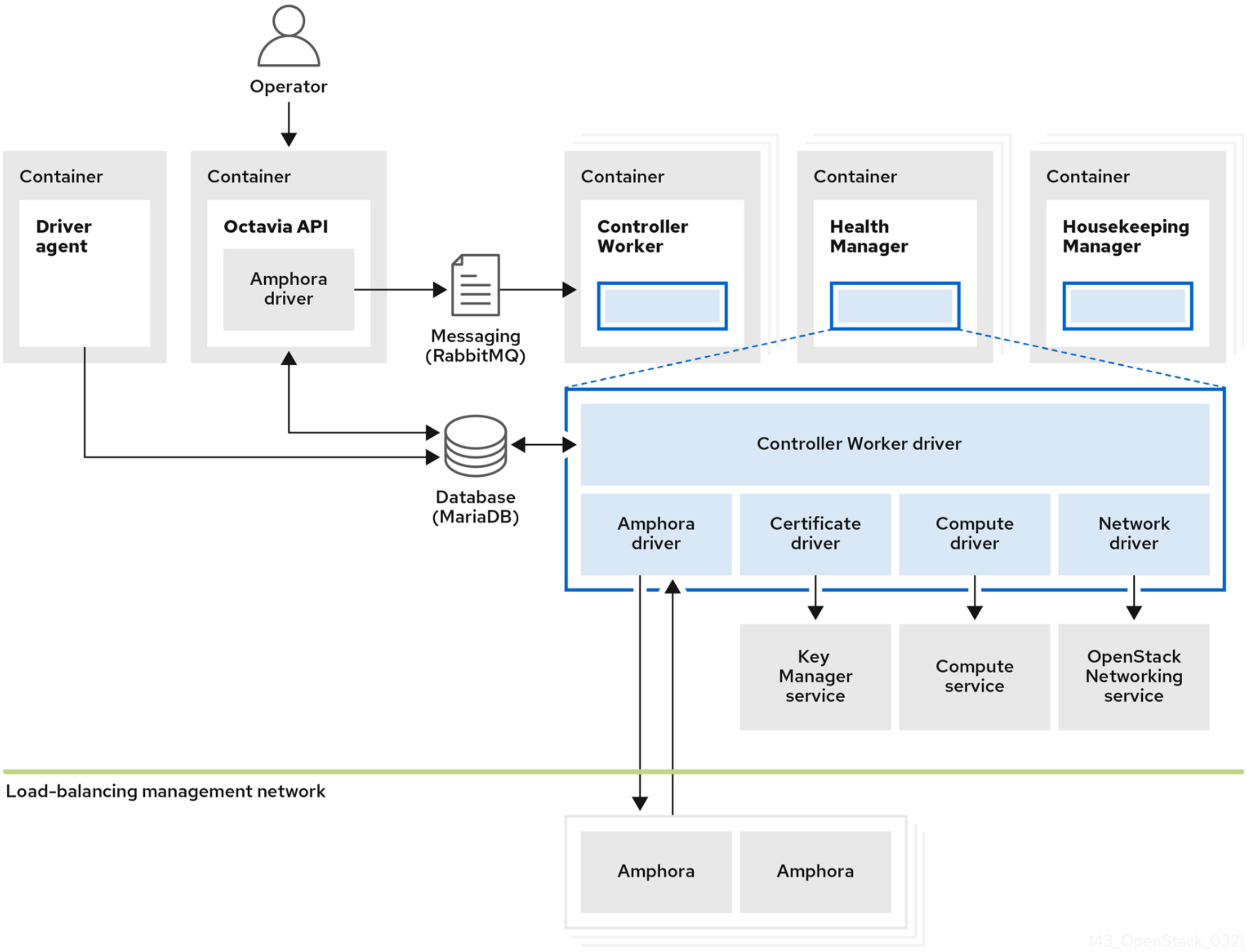


Figure 1.1 shows the components of the Load-balancing service are hosted on the same nodes as the Networking API server, which by default, is on the Controller nodes. The Load-balancing service consists of the following components:

Octavia API (**octavia_api** container)

Provides the REST API for users to interact with octavia.

Controller Worker (**octavia_worker** container)

Sends configuration and configuration updates to amphorae over the load-balancing management network.

Health Manager (**octavia_health_manager** container)

Monitors the health of individual amphorae and handles failover events if an amphora encounters a failure.

Housekeeping Manager (**octavia_housekeeping** container)

Cleans up deleted database records, and manages amphora certificate rotation.

Driver agent (**octavia_driver_agent** container)

Supports other provider drivers, such as OVN.

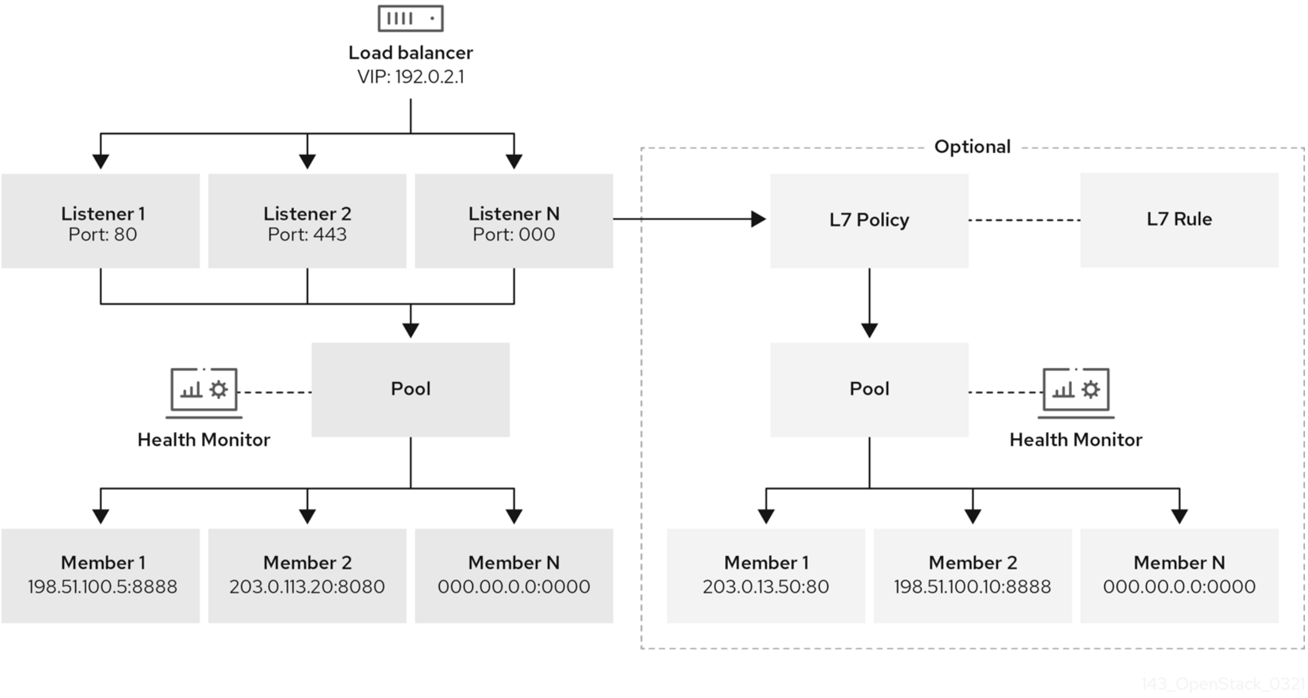
Amphora

Performs the load balancing. Amphorae are typically instances that run on Compute nodes that you configure with load balancing parameters according to the listener, pool, health monitor, L7 policies, and members' configuration. Amphorae send a periodic heartbeat to the Health Manager.

1.2. LOAD-BALANCING SERVICE OBJECT MODEL

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) uses a typical load-balancing object model.

Figure 1.2. Load-balancing service object model diagram



Load balancer

The top API object that represents the load-balancing entity. The VIP address is allocated when you create the load balancer. When you use the amphora provider to create the load balancer one or more amphora instances launch on one or more Compute nodes.

Listener

The port on which the load balancer listens, for example, TCP port 80 for HTTP.

Health Monitor

A process that performs periodic health checks on each back-end member server to pre-emptively detect failed servers and temporarily remove them from the pool.

Pool

A group of members that handle client requests from the load balancer. You can associate pools with more than one listener by using the API. You can share pools with L7 policies.

Member

Describes how to connect to the back-end instances or services. This description consists of the IP address and network port on which the back end member is available.

L7 Rule

Defines the layer 7 (L7) conditions that determine whether an L7 policy applies to the connection.

L7 Policy

A collection of L7 rules associated with a listener, and which might also have an association to a back-end pool. Policies describe actions that the load balancer takes if all of the rules in the policy are true.

Additional resources

- Section 1.1, “Load-balancing service components”

1.3. USES OF LOAD BALANCING IN RED HAT OPENSTACK PLATFORM

Load balancing is essential for enabling simple or automatic delivery scaling and availability for cloud deployments. The Load-balancing service (octavia) depends on other Red Hat OpenStack Platform (RHOSP) services:

- **Compute service (nova)** - For managing the Load-balancing service VM instance (amphora) lifecycle, and creating compute resources on demand.
- **Networking service (neutron)** - For network connectivity between amphorae, tenant environments, and external networks.
- **Key Manager service (barbican)** - For managing TLS certificates and credentials, when TLS session termination is configured on a listener.
- **Identity service (keystone)** - For authentication requests to the octavia API, and for the Load-balancing service to authenticate with other RHOSP services.
- **Image service (glance)** - For storing the amphora virtual machine image.
- **Common Libraries (oslo)** - For communication between Load-balancing service controller components, making Load-balancing service work within the standard OpenStack framework and review system, and project code structure.
- **Taskflow** - Is part of Common Libraries; the Load-balancing service uses this job flow system when it orchestrates back-end service configuration and management.

The Load-balancing service interacts with the other RHOSP services through a driver interface. The driver interface avoids major restructuring of the Load-balancing service if an external component requires replacement with a functionally-equivalent service.

CHAPTER 2. CONSIDERATIONS FOR IMPLEMENTING THE LOAD-BALANCING SERVICE

You must make several decisions when you plan to deploy the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) such as choosing which provider to use or whether to implement a highly available environment:

- Section 2.1, "Load-balancing service provider drivers"
- Section 2.2, "Load-balancing service (octavia) feature support matrix"
- Section 2.3, "Load-balancing service software requirements"
- Section 2.4, "Load-balancing service prerequisites for the undercloud"
- Section 2.5, "Basics of active-standby topology for Load-balancing service instances"
- Section 2.6, "Post-deployment steps for the Load-balancing service"

2.1. LOAD-BALANCING SERVICE PROVIDER DRIVERS

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) supports enabling multiple provider drivers by using the Octavia v2 API. You can choose to use one provider driver, or multiple provider drivers simultaneously.

RHOSP provides two load-balancing providers, amphora and Open Virtual Network (OVN).

Amphora, the default, is a highly available load balancer with a feature set that scales with your compute environment. Because of this, amphora is suited for large-scale deployments.

The OVN load-balancing provider is a lightweight load balancer with a basic feature set. OVN is typical for east-west, layer 4 network traffic. OVN provisions quickly and consumes fewer resources than a full-featured load-balancing provider such as amphora.

On RHOSP deployments that use the neutron Modular Layer 2 plug-in with the OVN mechanism driver (ML2/OVN), RHOSP director automatically enables the OVN provider driver in the Load-balancing service without the need for additional installation or configuration.

IMPORTANT

The information in this section applies only to the amphora load-balancing provider, unless indicated otherwise.

Additional resources

- Section 2.2, "Load-balancing service (octavia) feature support matrix"

2.2. LOAD-BALANCING SERVICE (OCTAVIA) FEATURE SUPPORT MATRIX

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) provides two load-balancing providers, amphora and Open Virtual Network (OVN).

Amphora is a full-featured load-balancing provider that requires a separate haproxy VM and an extra latency hop.

OVN runs on every node and does not require a separate VM nor an extra hop. However, OVN has far fewer load-balancing features than amphora.

The following table lists features in the Load-balancing service that Red Hat OpenStack Platform (RHOSP) 17.1 supports and in which maintenance release support for the feature began.

NOTE

If the feature is not listed, then RHOSP 17.1 does not support the feature.

Table 2.1. Load-balancing service (octavia) feature support matrix

Feature	Support level in RHOSP 17.1	
	Amphora Provider	**OVN Provider**
ML2/OVS L3 HA	Full support	No support
ML2/OVS DVR	Full support	No support
ML2/OVS L3 HA + composable network node [1]	Full support	No support
ML2/OVS DVR + composable network node [1]	Full support	No support
ML2/OVN L3 HA	Full support	Full support
ML2/OVN DVR	Full support	Full support
DPDK	No support	No support
SR-IOV	No support	No support
Health monitors	Full support	No support
Amphora active-standby	Full support	No support
Terminated HTTPS load balancers (with barbican)	Full support	No support
Amphora spare pool	Technology Preview only	No support
UDP	Full support	Full support
Backup members	Technology Preview only	No support

Provider framework	Technology Preview only	No support
TLS client authentication	Technology Preview only	No support
TLS back end encryption	Technology Preview only	No support
Octavia flavors	Full support	No support
Object tags	Full support	No support
Listener API timeouts	Full support	No support
Log offloading	Full support	No support
VIP access control list	Full support	No support
Availabilty zones	Full support	No support
Volume-based amphora	No support	No support

[1] Network node with OVS, metadata, DHCP, L3, and Octavia (worker, health monitor, and housekeeping).

Additional resources

- Section 2.1, “Load-balancing service provider drivers”

2.3. LOAD-BALANCING SERVICE SOFTWARE REQUIREMENTS

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) requires that you configure the following core OpenStack components:

- Compute (nova)
- OpenStack Networking (neutron)
- Image (glance)
- Identity (keystone)
- RabbitMQ
- MySQL

2.4. LOAD-BALANCING SERVICE PREREQUISITES FOR THE UNDERCLOUD

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) has the following requirements for the RHOSP undercloud:

- A successful undercloud installation.
- The Load-balancing service present on the undercloud.
- A container-based overcloud deployment plan.
- Load-balancing service components configured on your Controller nodes.

IMPORTANT

If you want to enable the Load-balancing service on an existing overcloud deployment, you must prepare the undercloud. Failure to do so results in the overcloud installation being reported as successful yet without the Load-balancing service running.

2.5. BASICS OF ACTIVE-STANDBY TOPOLOGY FOR LOAD-BALANCING SERVICE INSTANCES

When you deploy the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia), you can decide whether, by default, load balancers are highly available when users create them. If you want to give users a choice, then after RHOSP deployment, create a Load-balancing service flavor for creating highly available load balancers and a flavor for creating standalone load balancers.

By default, the amphora provider driver is configured for a single Load-balancing service (amphora) instance topology with limited support for high availability (HA). However, you can make Load-balancing service instances highly available when you implement an active-standby topology.

In this topology, the Load-balancing service boots an active and standby instance for each load balancer, and maintains session persistence between each. If the active instance becomes unhealthy, the instance automatically fails over to the standby instance, making it active. The Load-balancing service health manager automatically rebuilds an instance that fails.

Additional resources

- Section 4.2, “Enabling active-standby topology for Load-balancing service instances”

2.6. POST-DEPLOYMENT STEPS FOR THE LOAD-BALANCING SERVICE

Red Hat OpenStack Platform (RHOSP) provides a workflow task to simplify the post-deployment steps for the Load-balancing service (octavia). This workflow runs a set of Ansible playbooks to provide the following post-deployment steps as the last phase of the overcloud deployment:

- Configure certificates and keys.
- Configure the load-balancing management network between the amphorae and the Load-balancing service Controller worker and health manager.

Amphora image

On pre-provisioned servers, you must install the amphora image on the undercloud before you deploy the Load-balancing service:

```
$ sudo dnf install octavia-amphora-image-x86_64.noarch
```

On servers that are not pre-provisioned, RHOSP director automatically downloads the default amphora image, uploads it to the overcloud Image service (glance), and then configures the Load-balancing service to use this amphora image. During a stack update or upgrade, director updates this image to the latest amphora image.

NOTE

Custom amphora images are not supported.

Additional resources

- Section 4.1, “Deploying the Load-balancing service”

CHAPTER 3. SECURING THE LOAD-BALANCING SERVICE

To secure communication between the various components of the Red Hat OpenStack Load-balancing service (octavia) uses TLS encryption protocol and public key cryptography.

- Section 3.1, “Two-way TLS authentication in the Load-balancing service”
- Section 3.2, “Certificate lifecycles for the Load-balancing service”
- Section 3.3, “Configuring Load-balancing service certificates and keys”

3.1. TWO-WAY TLS AUTHENTICATION IN THE LOAD-BALANCING SERVICE

The controller processes of the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) communicate with Load-balancing service instances (amphorae) over a TLS connection. The Load-balancing service validates that both sides are trusted by using two-way TLS authentication.

NOTE

This is a simplification of the full TLS handshake process. For more information about the TLS handshake process, see TLS 1.3 RFC 8446.

There are two phases involved in two-way TLS authentication. In *Phase one* a Controller process, such as the Load-balancing service worker process, connects to a Load-balancing service instance, and the instance presents its server certificate to the Controller. The Controller then validates the server certificate against the server Certificate Authority (CA) certificate stored on the Controller. If the presented certificate is validated against the server CA certificate, the connection proceeds to phase two.

In *Phase two* the Controller presents its client certificate to the Load-balancing service instance. The instance then validates the certificate against the client CA certificate stored inside the instance. If this certificate is successfully validated, the rest of the TLS handshake continues to establish the secure communication channel between the Controller and the Load-balancing service instance.

Additional resources

- Section 3.3, “Configuring Load-balancing service certificates and keys”

3.2. CERTIFICATE LIFECYCLES FOR THE LOAD-BALANCING SERVICE

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) controller uses the server certificate authority certificates and keys to uniquely generate a certificate for each Load-balancing service instance (amphora).

The Load-balancing service housekeeping controller process automatically rotates these server certificates as they near their expiration date.

The Load-balancing service controller processes use the client certificates. The human operator who manages these TLS certificates usually grants a long expiration period because the certificates are used on the cloud control plane.

Additional resources

- Section 3.3, “Configuring Load-balancing service certificates and keys”

3.3. CONFIGURING LOAD-BALANCING SERVICE CERTIFICATES AND KEYS

You can configure Red Hat OpenStack Platform (RHOSP) director to generate certificates and keys, or you can supply your own. Configure director to automatically create the required private certificate authorities and issue the necessary certificates. These certificates are for internal Load-balancing service (octavia) communication only and are not exposed to users.

IMPORTANT

RHOSP director generates certificates and keys and automatically renews them before they expire. If you use your own certificates, you must remember to renew them.

NOTE

Switching from manually generated certificates to automatically generated certificates is not supported by RHOSP director. But you can enforce the re-creation of the certificates by deleting the existing certificates on the Controller nodes in the **/var/lib/config-data/puppet-generated/octavia/etc/octavia/certs** directory and updating the overcloud.

If you must use your own certificates and keys, then complete the following steps:

Prerequisites

- Read and understand, "Changing Load-balancing service default settings." (See link in "Additional resources.")

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Create a YAML custom environment file.

   Example

   ```
   $ vi /home/stack/templates/my-octavia-environment.yaml
   ```

4. In the YAML environment file, add the following parameters with values appropriate for your site:

   - **OctaviaCaCert**:
     The certificate for the CA that Octavia uses to generate certificates.

   - **OctaviaCaKey**:
     The private CA key used to sign the generated certificates.

   - **OctaviaCaKeyPassphrase**:

The passphrase used with the private CA key above.

- **OctaviaClientCert**:
  The client certificate and un-encrypted key issued by the Octavia CA for the controllers.

- **OctaviaGenerateCerts**:
  The Boolean that instructs director to enable (true) or disable (false) automatic certificate and key generation.

IMPORTANT

You must set **OctaviaGenerateCerts** to false.

Example

```
parameter_defaults:
    OctaviaCaCert: |
      -----BEGIN CERTIFICATE-----

MIIDgzCCAmugAwIBAgIJAKk46qw6ncJaMA0GCSqGSIb3DQEBCwUAMFgxCzAJBgNV
      [snip]
      sFW3S2roS4X0Af/kSSD8mlBBTFTCMBAj6rtLBKLaQbIxEpIzrgvp
      -----END CERTIFICATE-----

    OctaviaCaKey: |
      -----BEGIN RSA PRIVATE KEY-----
      Proc-Type: 4,ENCRYPTED
      [snip]
      -----END RSA PRIVATE KEY-----[

    OctaviaClientCert: |
      -----BEGIN CERTIFICATE-----

MIIDmjCCAoKgAwIBAgIBATANBgkqhkiG9w0BAQsFADBcMQswCQYDVQQGEwJVUzEP

      [snip]
      270l5ILSnfejLxDH+vI=
      -----END CERTIFICATE-----
      -----BEGIN PRIVATE KEY-----

MIIEvgIBADANBgkqhkiG9w0BAQEFAASCBKgwggSkAgEAAoIBAQDU771O8MTQV8RY

      [snip]
      KfrjE3UqTF+ZaaIQaz3yayXW
      -----END PRIVATE KEY-----

    OctaviaCaKeyPassphrase:
      b28c519a-5880-4e5e-89bf-c042fc75225d

    OctaviaGenerateCerts: false
    [rest of file snipped]
```

5. Run the **openstack overcloud deploy** command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
-e /home/stack/templates/my-octavia-environment.yaml
```

Additional resources

- Section 4.3, “Changing Load-balancing service default settings”
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

CHAPTER 4. INSTALLING AND CONFIGURING THE LOAD-BALANCING SERVICE

When you deploy the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) using RHOSP director, you can decide to make its VM instances (amphorae) highly available. You also use director when you want to make configuration changes to the Load-balancing service.

- Section 4.1, “Deploying the Load-balancing service”
- Section 4.2, “Enabling active-standby topology for Load-balancing service instances”
- Section 4.3, “Changing Load-balancing service default settings”

4.1. DEPLOYING THE LOAD-BALANCING SERVICE

You use Red Hat OpenStack Platform (RHOSP) director to deploy the Load-balancing service (octavia). Director uses Orchestration service (heat) templates that are a set of plans for your environment. The undercloud imports these plans and follows their instructions to create the Load-balancing service and your RHOSP environment.

Prerequisites

- Ensure that your environment has access to the octavia image.

Procedure

- Run the deployment command and include the core heat templates, environment files, and the **octavia.yaml** heat template.

  Example

  ```
  $ openstack overcloud deploy --templates \
  -e <your_environment_files> \
  -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml
  ```

  NOTE

  Director updates the amphorae to the latest amphora image during a stack update or upgrade.

Additional resources

- Deployment command options in the *Installing and managing Red Hat OpenStack Platform with director* guide

4.2. ENABLING ACTIVE-STANDBY TOPOLOGY FOR LOAD-BALANCING SERVICE INSTANCES

You can make Load-balancing service instances (amphorae) highly available when you implement an active-standby topology using Red Hat OpenStack Platform (RHOSP) director. Director uses Orchestration service (heat) templates that are a set of plans for your environment. The undercloud imports these plans and follows their instructions to create the Load-balancing service and your RHOSP environment.

Prerequisites

- Ensure that anti-affinity is enabled for the Compute service. This is the default.
- A minimum of three Compute node hosts:
  - Two Compute node hosts to place the amphorae on different hosts (Compute anti-affinity).
  - A third host to successfully fail over an active-standby load balancer, when a problem arises.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Create a custom YAML environment file.

   #### Example

   ```
   $ vi /home/stack/templates/my-octavia-environment.yaml
   ```

4. In the custom environment file, add the following parameter:

   ```
   parameter_defaults:
      OctaviaLoadBalancerTopology: "ACTIVE_STANDBY"
   ```

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   #### IMPORTANT

   The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

   #### Example

   ```
   $ openstack overcloud deploy --templates \
   -e <your_environment_files> \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
   -e /home/stack/templates/my-octavia-environment.yaml
   ```

Verification

- After the deployment is complete and you have created a load balancer, run the following commands:

```
$ source overcloudrc
$ openstack loadbalancer amphora list
```

If your Load-balancing service instance highly available configuration is successful, you see output for two instances (amphorae), and no occurrence of **role** equaling **SINGLE**.

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

4.3. CHANGING LOAD-BALANCING SERVICE DEFAULT SETTINGS

You make configuration changes to the Load-balancing service (octavia) using Red Hat OpenStack Platform (RHOSP) director. Director uses Orchestration service (heat) templates that are a set of plans for your environment. The undercloud imports these plans and follows their instructions to create the Load-balancing service and your RHOSP environment.

Prerequisites

- Determine which RHOSP Orchestration service (heat) parameters that director already uses to deploy the Load-balancing service by consulting the following file on the undercloud:

```
/usr/share/openstack-tripleo-heat-templates/deployment/octavia/octavia-deployment-config.j2.yaml
```

- Decide which parameters that you want to modify.
  Here are a few examples:
  - **OctaviaControlNetwork**
    The name of the neutron network used for the load-balancing management network.
  - **OctaviaControlSubnetCidr**
    The subnet for amphora control subnet in CIDR form.

    You can use this parameter to specify the amphora control subnet as an IPv6 CIDR. Using an IPv6 CIDR in **OctaviaControlSubnetCidr** requires you to also set IPv6 addresses in **OctaviaControlSubnetPoolStart** and **OctaviaControlSubnetPoolEnd**.

    IMPORTANT

    This feature, using IPv6 CIDRs for the load-balancing management network, is available in this release as a *Technology Preview*, and therefore is not fully supported by Red Hat. It should only be used for testing, and should not be deployed in a production environment.

    For more information about Technology Preview features, see Scope of Coverage Details.

> **IMPORTANT**
>
> After you have set CIDRs for the amphora control subnet, you cannot modify these values. This also means that you cannot upgrade from an IPv4 subnet to an IPv6 subnet.

- **OctaviaMgmtPortDevName**
  The name of the octavia management network interface used for communication between octavia worker/health-manager with the amphora machine.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Create a custom YAML environment file.

   Example

   ```
   $ vi /home/stack/templates/my-octavia-environment.yaml
   ```

4. Your environment file must contain the keywords **parameter_defaults**. Put your parameter value pairs after the **parameter_defaults** keyword.

   Example

   ```
   parameter_defaults:
     OctaviaMgmtPortDevName: "o-hm0"
     OctaviaControlNetwork: 'lb-mgmt-net'
     OctaviaControlSubnet: 'lb-mgmt-subnet'
     OctaviaControlSecurityGroup: 'lb-mgmt-sec-group'
     OctaviaControlSubnetCidr: '172.24.0.0/16'
     OctaviaControlSubnetGateway: '172.24.0.1'
     OctaviaControlSubnetPoolStart: '172.24.0.2'
     OctaviaControlSubnetPoolEnd: '172.24.255.254'
   ```

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   > **IMPORTANT**
   >
   > The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

   Example

   ```
   $ openstack overcloud deploy --templates \
   -e <your_environment_files> \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
   -e /home/stack/templates/my-octavia-environment.yaml
   ```

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

CHAPTER 5. MANAGING LOAD-BALANCING SERVICE INSTANCE LOGS

You can enable tenant flow logging or suppress logging to the amphora local file system. You can also forward administrative or tenant flow logs to syslog receivers in a set of containers or to other syslog receivers at endpoints of your choosing.

When you choose to use the TCP syslog protocol, you can specify one or more secondary endpoints for administrative and tenant log offloading in the event that the primary endpoint fails.

In addition, you can control a range of other logging features such as, setting the syslog facility value, changing the tenant flow log format, and widening the scope of administrative logging to include logs from sources like the kernel and from cron.

- Section 5.1, “Enabling Load-balancing service instance administrative log offloading”
- Section 5.2, “Enabling tenant flow log offloading for Load-balancing service instances”
- Section 5.3, “Disabling Load-balancing service instance tenant flow logging”
- Section 5.4, “Disabling Load-balancing service instance local log storage”
- Section 5.5, “Heat parameters for Load-balancing service instance logging”
- Section 5.6, “Load-balancing service instance tenant flow log format”

5.1. ENABLING LOAD-BALANCING SERVICE INSTANCE ADMINISTRATIVE LOG OFFLOADING

By default, Load-balancing service instances (amphorae) store logs on the local machine in the systemd journal. However, you can specify that the amphorae offload logs to syslog receivers to aggregate administrative logs. Log offloading enables administrators to go to one location for logs, and retain logs when the amphorae are rotated.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. Create a custom YAML environment file.

Example

```
$ vi /home/stack/templates/my-octavia-environment.yaml
```

4. In the YAML environment file under **parameter_defaults**, set **OctaviaLogOffload** to **true**.

```
parameter_defaults:
   OctaviaLogOffload: true
   ...
```

> NOTE
>
> The amphorae offload administrative logs use the syslog facility value of **local1**, by default, unless you specify another value with the **OctaviaAdminLogFacility** parameter. The valid values are 0 – 7.
>
> **Example**
>
> ```
> parameter_defaults:
>   OctaviaLogOffload: true
>   OctaviaAdminLogFacility: 2
>   ...
> ```

5. The amphorae forward only load balancer-related administrative logs, such as the haproxy admin logs, keepalived, and amphora agent logs. If you want to configure the amphorae to send all of the administrative logs from the amphorae, such as the kernel, system, and security logs, set **OctaviaForwardAllLogs** to **true**.

   Example

   ```
   parameter_defaults:
     OctaviaLogOffload: true
     OctaviaForwardAllLogs: true
     ...
   ```

6. Choose the log protocol: UDP (default) or TCP.
   In the event that the primary endpoint fails, the amphorae sends its logs to a secondary endpoint only when the log protocol is TCP.

   ```
   parameter_defaults:
     OctaviaLogOffload: true
     OctaviaConnectionLogging: false
     OctaviaLogOffloadProtocol: tcp
     ...
   ```

7. The amphorae use a set of default containers defined by the Orchestration service (heat) that contain syslog receivers listening for log messages. If you want to use a different set of endpoints, you can specify those with the **OctaviaAdminLogTargets** parameter.
   The endpoints configured for tenant flow log offloading can be the same endpoints used for administrative log offloading.

   Also, if your log offload protocol is TCP, in the event that the first endpoint is unreachable, the amphorae will try the additional endpoints in the order that you list them until a connection succeeds.

   ```
   OctaviaAdminLogTargets: <ip_address>:<port>[, <ip_address>:<port>]
   ```

   Example

   ```
   parameter_defaults:
     OctaviaLogOffload: true
     OctaviaLogOffloadProtocol: tcp
   ```

```
  OctaviaAdminLogTargets: 192.0.2.1:10514, 2001:db8:1::10:10514
  ...
```

8. By default, when you enable log offloading, tenant flow logs are also offloaded.
   If you want to disable tenant flow log offloading, set the **OctaviaConnectionLogging** to **false**.

   Example

   ```
   parameter_defaults:
     OctaviaLogOffload: true
     OctaviaConnectionLogging: false
     ...
   ```

9. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

   IMPORTANT

   The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

   Example

   ```
   $ openstack overcloud deploy --templates \
   -e [your-environment-files] \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
   -e /home/stack/templates/my-octavia-environment.yaml
   ```

Verification

- Unless you specified specific endpoints with the **OctaviaAdminLogTargets** or **OctaviaTenantLogTargets**, the amphorae offload logs to the RHOSP Controller in the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**).
- Check the appropriate location for the presence of the following log files:
  - **octavia-amphora.log**-- Log file for the administrative log.
  - (if enabled) **octavia-tenant-traffic.log**-- Log file for the tenant traffic flow log.

Additional resources

- Section 5.5, "Heat parameters for Load-balancing service instance logging"
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

5.2. ENABLING TENANT FLOW LOG OFFLOADING FOR LOAD-BALANCING SERVICE INSTANCES

By default, Load-balancing service instances (amphorae) store logs on the local machine in the systemd journal. However, you can specify alternate syslog receiver endpoints instead. Because tenant flow logs grow in size depending on the number of tenant connections, ensure that the alternate syslog receivers contain sufficient disk space.

By default, tenant flow log offloading is automatically enabled when administrative log offloading is enabled. To turn off tenant flow log offloading, set the **OctaviaConnectionLogging** parameter to **false**.

IMPORTANT

Tenant flow logging can produce a large number of syslog messages depending on how many connections the load balancers are receiving. Tenant flow logging produces one log entry for each connection to the load balancer. Monitor log volume and configure your syslog receivers appropriately based on the expected number of connections that your load balancers manage.

Procedure

1. Log in to the undercloud host as the **stack** user.
2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Locate the environment file in which the **OctaviaConnectionLogging** parameter is set:

   ```
   $ grep -rl OctaviaConnectionLogging /home/stack/templates/
   ```

4. If you do not find the file, create an environment file:

   ```
   $ vi /home/stack/templates/my-octavia-environment.yaml
   ```

5. Add the **OctaviaLogOffload** and **OctaviaConnectionLogging** parameters to the **parameter_defaults** section of the environment file and set the values to **true**:

   ```
   parameter_defaults:
      OctaviaLogOffload: true
      OctaviaConnectionLogging: true
      ...
   ```

   NOTE

   The amphorae use the syslog facility default value of **local0** to offload tenant flow logs unless you use the **OctaviaTenantLogFacility** parameter to specify another value. The valid values are 0 – 7.

6. Optional: To change the default endpoint that the amphorae use for both tenant and administrative log offloading, use **OctaviaTenantLogTargets** and **OctaviaAdminLogTargets**, respectively. The amphorae use a set of default containers that contain syslog receivers that listen for log messages.
   Also, if your log offload protocol is TCP, in the event that the first endpoint is unreachable, the amphorae will try the additional endpoints in the order that you list them until a connection succeeds.

```
OctaviaAdminLogTargets: <ip-address>:<port>[, <ip-address>:<port>]
OctaviaTenantLogTargets: <ip-address>:<port>[, <ip-address>:<port>]
```

7. Run the deployment command and include the core heat templates, environment files, and the custom environment file you modified.

IMPORTANT

The order of the environment files is important because the parameters and resources defined in subsequent environment files take precedence.

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
-e /home/stack/templates/my-octavia-environment.yaml
```

Verification

- Unless you specified specific endpoints with the **OctaviaAdminLogTargets** or **OctaviaTenantLogTargets**, the amphorae offload logs to the RHOSP Controller in the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**).
- Check the appropriate location for the presence of the following log files:
  - **octavia-amphora.log**-- Log file for the administrative log.
  - **octavia-tenant-traffic.log**-- Log file for the tenant traffic flow log.

Additional resources

- Section 5.5, “Heat parameters for Load-balancing service instance logging”
- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Section 5.6, “Load-balancing service instance tenant flow log format”

5.3. DISABLING LOAD-BALANCING SERVICE INSTANCE TENANT FLOW LOGGING

Tenant flow log offloading for Load-balancing service instances (amphorae) is automatically enabled when you enable administrative log offloading.

To keep administrative log offloading enabled and to disable tenant flow logging, you must set the **OctaviaConnectionLogging** parameter to **false**.

When the **OctaviaConnectionLogging** parameter is **false**, the amphorae do not write tenant flow logs to the disk inside the amphorae, nor offload any logs to syslog receivers listening elsewhere.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

   ```
   $ source ~/stackrc
   ```

3. Locate the YAML custom environment file in which amphora logging is configured.

   ### Example

   ```
   $ grep -rl OctaviaLogOffload /home/stack/templates/
   ```

4. In the custom environment file, under **parameter_defaults**, set **OctaviaConnectionLogging** to **false**.

   ### Example

   ```
   parameter_defaults:
      OctaviaLogOffload: true
      OctaviaConnectionLogging: false
      ...
   ```

5. Run the deployment command and include the core heat templates, environment files, and the custom environment file in which you set **OctaviaConnectionLogging** to **true**.

   IMPORTANT

   The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

   ### Example

   ```
   $ openstack overcloud deploy --templates \
   -e [your-environment-files] \
   -e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
   -e /home/stack/templates/my-octavia-environment.yaml
   ```

Verification

- Unless you specified specific endpoints with the **OctaviaAdminLogTargets** or **OctaviaTenantLogTargets**, the amphorae offload logs to the RHOSP Controller in the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**).

- Check the appropriate location for the *absence* of **octavia-tenant-traffic.log**.

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide

- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

5.4. DISABLING LOAD-BALANCING SERVICE INSTANCE LOCAL LOG STORAGE

Even when you configure Load-balancing service instances (amphorae) to offload administrative and tenant flow logs, the amphorae continue to write these logs to the disk inside the amphorae. To improve the performance of the load balancer, you can stop logging locally.

IMPORTANT

If you disable logging locally, you also disable all log storage in the amphora, including kernel, system, and security logging.

NOTE

If you disable local log storage and the **OctaviaLogOffload** parameter is set to false, ensure that you set **OctaviaConnectionLogging** to false for improved load balancing performance.

Procedure

1. Log in to the undercloud host as the **stack** user.

2. Source the undercloud credentials file:

```
$ source ~/stackrc
```

3. Create a custom YAML environment file.

Example

```
$ vi /home/stack/templates/my-octavia-environment.yaml
```

4. In the environment file under **parameter_defaults**, set **OctaviaDisableLocalLogStorage** to **true**.

```
parameter_defaults:
    OctaviaDisableLocalLogStorage: true
    ...
```

5. Run the deployment command and include the core heat templates, environment files, and this new custom environment file.

IMPORTANT

The order of the environment files is important as the parameters and resources defined in subsequent environment files take precedence.

Example

```
$ openstack overcloud deploy --templates \
-e <your_environment_files> \
-e /usr/share/openstack-tripleo-heat-templates/environments/services/octavia.yaml \
-e /home/stack/templates/my-octavia-environment.yaml
```

Verification

- On the amphora instance, check the location where log files are written, and verify that no new log files are being written.

Additional resources

- Environment files in the *Customizing your Red Hat OpenStack Platform deployment* guide
- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

5.5. HEAT PARAMETERS FOR LOAD-BALANCING SERVICE INSTANCE LOGGING

When you want to configure Load-balancing service instance (amphora) logging, you set values for one or more Orchestration service (heat) parameters that control logging and run the **openstack overcloud deploy** command.

These heat parameters for amphora logging enable you to control features such as turning on log offloading, defining custom endpoints to offload logs to, setting the syslog facility value for logs, and so on.

Table 5.1. Heat parameters for all logs

Parameter	Default	Description
OctaviaLogOffload	**false**	When **true**, instances offload their logs. If no endpoints are specified, then, by default, the instance offloads its log to the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**).
OctaviaDisableLocalLogStorage	**false**	When **true**, instances do not store logs on the instance host filesystem. This includes all kernel, system, and security logs.
OctaviaForwardAllLogs	**false**	When **true**, instances forward all log messages to the administrative log endpoints, including non-load balancing related logs such as the cron and kernel logs. For instances to recognize **OctaviaForwardAllLogs**, you must also enable **OctaviaLogOffload**.

Table 5.2. Heat parameters for admin logging

Parameter	Default	Description

Parameter	Default	Description
OctaviaAdminLogTargets	No value.	A comma-delimited list of syslog endpoints (<host>:<port>) to receive administrative log messages. These endpoints can be a container, VM, or physical host that is running a process that is listening for the log messages on the specified port. When **OctaviaAdminLogTargets** is not present, the instance offloads its log to the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**) on a set of containers that is defined by RHOSP director.
OctaviaAdminLogFacility	**1**	A number between 0 and 7 that is the syslog "LOG_LOCAL" facility to use for the administrative log messages.

Table 5.3. Heat parameters for tenant flow logging

Parameter	Default	Description
OctaviaConnectionLogging	**true**	When **true**, tenant connection flows are logged. When **OctaviaConnectionLogging** is false, the amphorae stop logging tenant connections regardless of the **OctaviaLogOffload** setting. **OctaviaConnectionLogging** disables local tenant flow log storage and, if log offloading is enabled, it does not forward tenant flow logs.
OctaviaTenantLogTargets	No value.	A comma-delimited list of syslog endpoints (<host>:<port>) to receive tenant traffic flow log messages. These endpoints can be a container, VM, or physical host that is running a process that is listening for the log messages on the specified port. When **OctaviaTenantLogTargets** is not present, the instance offloads its log to the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**) on a set of containers that is defined by RHOSP director.
OctaviaTenantLogFacility	**0**	A number between 0 and 7 that is the syslog "LOG_LOCAL" facility to use for the tenant traffic flow log messages.

Parameter	Default	Description
OctaviaUserLogFormat	**"{{ '{{' }} project_id {{ '}}' }} {{ '{{' }} lb_id {{ '}}' }} %f %ci %cp %t %{+Q}r %ST %B %U %[ssl_c_verify] %{+Q}[ssl_c_s_dn] %b %s %Tt %tsc"**	The format for the tenant traffic flow log. The alphanumerics represent specific octavia fields, and the curly braces ({}) are substitution variables.

Additional resources

- Including environment files in overcloud creation in the *Customizing your Red Hat OpenStack Platform deployment* guide

5.6. LOAD-BALANCING SERVICE INSTANCE TENANT FLOW LOG FORMAT

The log format that the tenant flow logs for Load-balancing service instances (amphorae) follows is the HAProxy log format. The two exceptions are the **project_id** and **lb_id** variables whose values are provided by the amphora provider driver.

Sample

Here is a sample log entry with rsyslog as the syslog receiver:

```
Jun 12 00:44:13 amphora-3e0239c3-5496-4215-b76c-6abbe18de573 haproxy[1644]:
5408b89aa45b48c69a53dca1aaec58db fd8f23df-960b-4b12-ba62-2b1dff661ee7 261ecfc2-9e8e-
4bba-9ec2-3c903459a895 172.24.4.1 41152 12/Jun/2019:00:44:13.030 "GET / HTTP/1.1" 200 76 73
- "" e37e0e04-68a3-435b-876c-cffe4f2138a4 6f2720b3-27dc-4496-9039-1aafe2fee105 4 --
```

Notes

- A hyphen (-) indicates any field that is unknown or not applicable to the connection.
- The prefix in the earlier sample log entry originates from the rsyslog receiver, and is not part of the syslog message from the amphora:

  ```
  Jun 12 00:44:13 amphora-3e0239c3-5496-4215-b76c-6abbe18de573 haproxy[1644]:”
  ```

Default

The default amphora tenant flow log format is:

```
`"{{ '{{' }} project_id {{ '}}' }} {{ '{{' }} lb_id {{ '}}' }} %f %ci %cp %t %{+Q}r %ST %B %U %[ssl_c_verify]
%{+Q}[ssl_c_s_dn] %b %s %Tt %tsc"`
```

Refer to the table that follows for a description of the format.

Table 5.4. Data variables for tenant flow logs format variable definitions.

Variable	Type	Field name
{{project_id}}	UUID	Project ID (substitution variable from the amphora provider driver)
{{lb_id}}	UUID	Load balancer ID (substitution variable from the amphora provider driver)
%f	string	**frontend_name**
%ci	IP address	**client_ip**
%cp	numeric	**client_port**
%t	date	**date_time**
%ST	numeric	**status_code**
%B	numeric	**bytes_read**
%U	numeric	**bytes_uploaded**
%ssl_c_verify	Boolean	**client_certificate_verify** (0 or 1)
%ssl_c_s_dn	string	**client_certificate_distinguised_name**
%b	string	**pool_id**
%s	string	**member_id**
%Tt	numeric	**processing_time** (milliseconds)
%tsc	string	**termination_state** (with cookie status)

Additional resources

- Custom log format in *HAProxy Documentation*

CHAPTER 6. CONFIGURING LOAD-BALANCING SERVICE FLAVORS

Load-balancing service (octavia) *flavors* are sets of provider configuration options that you create. When users request load balancers, they can specify that the load balancer be built using one of the defined flavors. You define a flavor for each load-balancing provider driver, which exposes the unique capabilities of the respective provider.

To create a new Load-balancing service flavor:

1. Decide which capabilities of the load-balancing provider you want to configure in the flavor.
2. Create the flavor profile with the flavor capabilities you have chosen.
3. Create the flavor.
   - Section 6.1, “Listing Load-balancing service provider capabilities”
   - Section 6.2, “Defining flavor profiles”
   - Section 6.3, “Creating Load-balancing service flavors”

6.1. LISTING LOAD-BALANCING SERVICE PROVIDER CAPABILITIES

You can review the list of capabilities that each Load-balancing service (octavia) provider driver exposes.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.

   Example

   ```
   $ source ~/my_overcloudrc
   ```

2. List the capabilities for each driver:

   ```
   $ openstack loadbalancer provider capability list <provider>
   ```

   Replace **<provider>** with the name or UUID of the provider.

   Example

   ```
   $ openstack loadbalancer provider capability list amphora
   ```

   The command output lists all of the capabilities that the provider supports.

   Sample output

   ```
   +-----------------------+-------------------------------------------------------+
   | name                  | description                                           |
   +-----------------------+-------------------------------------------------------+
   | loadbalancer_topology | The load balancer topology. One of: SINGLE - One  |
   ```

```
|                       | amphora per load balancer. ACTIVE_STANDBY - Two   |
|                       | amphora per load balancer.                        |
| ...                   | ...                                               |
+-----------------------+---------------------------------------------------+
```

3. Note the names of the capabilities that you want to include in the flavor that you are creating.

Additional resources

- Section 6.2, “Defining flavor profiles”
- loadbalancer provider capability list in the *Command line interface reference*

6.2. DEFINING FLAVOR PROFILES

Load-balancing service (octavia) flavor profiles contain the provider driver name and a list of capabilities. You use flavor profiles to create flavors that users specify to create a load balancer.

Prerequisites

- You must know which load-balancing provider and which of its capabilities you want to include in the flavor profile.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.

   Example

   ```
   $ source ~/my_overcloudrc
   ```

2. Create a flavor profile:

   ```
   $ openstack loadbalancer flavorprofile create --name <profile_name> --provider
   <provider_name> --flavor-data '{"<capability>": "<value>"}'
   ```

   Example

   ```
   $ openstack loadbalancer flavorprofile create --name amphora-single-profile --provider
   amphora --flavor-data '{"loadbalancer_topology": "SINGLE"}'
   ```

   Sample output

   ```
   +---------------+--------------------------------------+
   | Field         | Value                                |
   +---------------+--------------------------------------+
   | id            | 72b53ac2-b191-48eb-8f73-ed012caca23a |
   | name          | amphora-single-profile               |
   | provider_name | amphora                              |
   | flavor_data   | {"loadbalancer_topology": "SINGLE"}  |
   +---------------+--------------------------------------+
   ```

In this example, a flavor profile is created for the amphora provider. When this profile is specified in a flavor, the load balancer that users create by using the flavor is a single amphora load balancer.

Verification

- When you create a flavor profile, the Load-balancing service validates the flavor values with the provider to ensure that the provider can support the capabilities that you have specified.

Additional resources

- Section 6.3, "Creating Load-balancing service flavors"
- loadbalancer flavorprofile create in the *Command line interface reference*

6.3. CREATING LOAD-BALANCING SERVICE FLAVORS

You create a user-facing flavor for the Load-balancing service (octavia) by using a flavor profile. The name that you assign to the flavor is the value that users specify when they create a load balancer.

Prerequisites

- You must have created a flavor profile.

Procedure

1. Source a credentials file that gives you access to the overcloud with the RHOSP admin role.

   #### Example

   ```
   $ source ~/my_overcloudrc
   ```

2. Create a flavor:

   ```
   $ openstack loadbalancer flavor create --name <flavor_name> \
   --flavorprofile <flavor-profile> --description "<string>"
   ```

   #### TIP

   Provide a detailed description so that users can understand the capabilities of the flavor that you are providing.

   #### Example

   ```
   $ openstack loadbalancer flavor create --name standalone-lb --flavorprofile amphora-single-profile --description "A non-high availability load balancer for testing."
   ```

   #### Sample output

   ```
   +-------------------+--------------------------------------+
   | Field             | Value                                |
   +-------------------+--------------------------------------+
   ```

```
| id                | 25cda2d8-f735-4744-b936-d30405c05359 |
| name              | standalone-lb                        |
| flavor_profile_id | 72b53ac2-b191-48eb-8f73-ed012caca23a |
| enabled           | True                                 |
| description       | A non-high availability load         |
|                   | balancer for testing.                |
+-------------------+--------------------------------------+
```

In this example, a flavor has been defined. When users specify this flavor, they create a load balancer that uses one Load-balancing service instance (amphora) and is not highly available.

NOTE

Disabled flavors are still visible to users, but users cannot use the disabled flavor to create a load balancer.

Additional resources

- Section 6.2, “Defining flavor profiles”
- loadbalancer flavor create in the *Command line interface reference*

CHAPTER 7. MONITORING THE LOAD-BALANCING SERVICE

To keep load balancing operational, you can use the load-balancer management network and create, modify, and delete load-balancing health monitors.

7.1. LOAD-BALANCING MANAGEMENT NETWORK

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) monitors load balancers through a project network referred to as the *load-balancing management network* . Hosts that run the Load-balancing service must have interfaces to connect to the load-balancing management network. The supported interface configuration works with the neutron Modular Layer 2 plug-in with the Open Virtual Network mechanism driver (ML2/OVN) or the Open vSwitch mechanism driver (ML2/OVS). Use of the interfaces with other mechanism drivers has not been tested.

The default interfaces created at deployment are internal Open vSwitch (OVS) ports on the default integration bridge **br-int**. You must associate these interfaces with actual Networking service (neutron) ports allocated on the load-balancer management network.

The default interfaces are by default named, **o-hm0**. They are defined through standard interface configuration files on the Load-balancing service hosts. RHOSP director automatically configures a Networking service port and an interface for each Load-balancing service host during deployment. Port information and a template is used to create the interface configuration file, including:

- IP network address information including the IP and netmask
- MTU configuration
- the MAC address
- the Networking service port ID

In the default OVS case, the Networking service port ID is used to register extra data with the OVS port. The Networking service recognizes this interface as belonging to the port and configures OVS so it can communicate on the load-balancer management network.

By default, RHOSP configures security groups and firewall rules that allow the Load-balancing service controllers to communicate with its VM instances (amphorae) on TCP port 9443, and allows the heartbeat messages from the amphorae to arrive on the controllers on UDP port 5555. Different mechanism drivers might require additional or alternate requirements to allow communication between load-balancing services and the load balancers.

7.2. LOAD-BALANCING SERVICE INSTANCE MONITORING

The Load-balancing service (octavia) monitors the load balancing instances (amphorae) and initiates failovers and replacements if the amphorae malfunction. Any time a failover occurs, the Load-balancing service logs the failover in the corresponding health manager log on the controller in **/var/log/containers/octavia**.

Use log analytics to monitor failover trends to address problems early. Problems such as Networking service (neutron) connectivity issues, Denial of Service attacks, and Compute service (nova) malfunctions often lead to higher failover rates for load balancers.

7.3. LOAD-BALANCING SERVICE POOL MEMBER MONITORING

The Load-balancing service (octavia) uses the health information from the underlying load balancing subsystems to determine the health of members of the load-balancing pool. Health information is streamed to the Load-balancing service database, and made available by the status tree or other API methods. For critical applications, you must poll for health information in regular intervals.

7.4. LOAD BALANCER PROVISIONING STATUS MONITORING

You can monitor the provisioning status of a load balancer and send alerts if the provisioning status is **ERROR**. Do not configure an alert to trigger when an application is making regular changes to the pool and enters several **PENDING** stages.

The provisioning status of load balancer objects reflect the ability of the control plane to contact and successfully provision a create, update, and delete request. The operating status of a load balancer object reports on the current functionality of the load balancer.

For example, a load balancer might have a provisioning status of **ERROR**, but an operating status of **ONLINE**. This might be caused by a Networking (neutron) failure that blocked that last requested update to the load balancer configuration from successfully completing. In this case, the load balancer continues to process traffic through the load balancer, but might not have applied the latest configuration updates yet.

7.5. LOAD BALANCER FUNCTIONALITY MONITORING

You can monitor the operational status of your load balancer and its child objects.

You can also use an external monitoring service that connects to your load balancer listeners and monitors them from outside of the cloud. An external monitoring service indicates if there is a failure outside of the Load-balancing service (octavia) that might impact the functionality of your load balancer, such as router failures, network connectivity issues, and so on.

7.6. ABOUT LOAD-BALANCING SERVICE HEALTH MONITORS

A Load-balancing service (octavia) health monitor is a process that does periodic health checks on each back end member server to pre-emptively detect failed servers and temporarily pull them out of the pool.

If the health monitor detects a failed server, it removes the server from the pool and marks the member in **ERROR**. After you have corrected the server and it is functional again, the health monitor automatically changes the status of the member from **ERROR** to **ONLINE**, and resumes passing traffic to it.

Always use health monitors in production load balancers. If you do not have a health monitor, failed servers are not removed from the pool. This can lead to service disruption for web clients.

There are several types of health monitors, as briefly described here:

HTTP

by default, probes the / path on the application server.

HTTPS

operates exactly like HTTP health monitors, but with TLS back end servers.
If the servers perform client certificate validation, HAProxy does not have a valid certificate. In these cases, TLS-HELLO health monitoring is an alternative.

TLS-HELLO

ensures that the back end server responds to SSLv3-client hello messages.
A TLS-HELLO health monitor does not check any other health metrics, like status code or body contents.

PING

sends periodic ICMP ping requests to the back end servers.
You must configure back end servers to allow PINGs so that these health checks pass.

> IMPORTANT
>
> A PING health monitor checks only if the member is reachable and responds to ICMP echo requests. PING health monitors do not detect if the application that runs on an instance is healthy. Use PING health monitors only in cases where an ICMP echo request is a valid health check.

TCP

opens a TCP connection to the back end server protocol port.
The TCP application opens a TCP connection and, after the TCP handshake, closes the connection without sending any data.

UDP-CONNECT

performs a basic UDP port connect.
A UDP-CONNECT health monitor might not work correctly if Destination Unreachable (ICMP type 3) is not enabled on the member server, or if it is blocked by a security rule. In these cases, a member server might be marked as having an operating status of **ONLINE** when it is actually down.

7.7. CREATING LOAD-BALANCING SERVICE HEALTH MONITORS

Use Load-balancing service (octavia) health monitors to avoid service disruptions for your users. The health monitors run periodic health checks on each back end server to pre-emptively detect failed servers and temporarily pull the servers out of the pool.

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

2. Run the **openstack loadbalancer healthmonitor create** command, using argument values that are appropriate for your site.

   - All health monitor types require the following configurable arguments:

     <pool>

     Name or ID of the pool of back-end member servers to be monitored.

     --type

     The type of health monitor. One of **HTTP**, **HTTPS**, **PING**, **SCTP**, **TCP**, **TLS-HELLO**, or **UDP-CONNECT**.

     --delay

     Number of seconds to wait between health checks.

     --timeout

     Number of seconds to wait for any given health check to complete. **timeout** must always be smaller than **delay**.

     --max-retries

     Number of health checks a back-end server must fail before it is considered down. Also, the number of health checks that a failed back-end server must pass to be considered up again.

   - In addition, HTTP health monitor types also require the following arguments, which are set by default:

     --url-path

     Path part of the URL that should be retrieved from the back-end server. By default this is /.

     --http-method

     HTTP method that is used to retrieve the **url_path**. By default this is **GET**.

     --expected-codes

     List of HTTP status codes that indicate an OK health check. By default this is **200**.

     #### Example

     ```
     $ openstack loadbalancer healthmonitor create --name my-health-monitor --delay 10
     --max-retries 4 --timeout 5 --type TCP lb-pool-1
     ```

Verification

- Run the **openstack loadbalancer healthmonitor list** command and verify that your health monitor is running.

Additional resources

- loadbalancer healthmonitor create in the *Command line interface reference*

7.8. MODIFYING LOAD-BALANCING SERVICE HEALTH MONITORS

You can modify the configuration for Load-balancing service (octavia) health monitors when you want to change the interval for sending probes to members, the connection timeout interval, the HTTP method for requests, and so on.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Modify your health monitor (**my-health-monitor**).
   In this example, a user is changing the time in seconds that the health monitor waits between sending probes to members.

   Example

   ```
   $ openstack loadbalancer healthmonitor set my_health_monitor --delay 600
   ```

Verification

- Run the **openstack loadbalancer healthmonitor show** command to confirm your configuration changes.

  ```
  $ openstack loadbalancer healthmonitor show my_health_monitor
  ```

Additional resources

- loadbalancer healthmonitor set in the *Command line interface reference*
- loadbalancer healthmonitor show in the *Command line interface reference*

7.9. DELETING LOAD-BALANCING SERVICE HEALTH MONITORS

You can remove a Load-balancing service (octavia) health monitor.

TIP

An alternative to deleting a health monitor is to disable it by using the **openstack loadbalancer healthmonitor set --disable** command.

Procedure

1. Source your credentials file.

Example

```
$ source ~/overcloudrc
```

2. Delete the health monitor (**my-health-monitor**).

   Example

   ```
   $ openstack loadbalancer healthmonitor delete my-health-monitor
   ```

Verification

- Run the **openstack loadbalancer healthmonitor list** command to verify that the health monitor you deleted no longer exists.

Additional resources

- loadbalancer healthmonitor delete in the *Command line interface reference*

7.10. BEST PRACTICES FOR LOAD-BALANCING SERVICE HTTP HEALTH MONITORS

When you write the code that generates the health check in your web application, use the following best practices:

- The health monitor **url-path** does not require authentication to load.
- By default, the health monitor **url-path** returns an **HTTP 200 OK** status code to indicate a healthy server unless you specify alternate **expected-codes**.
- The health check does enough internal checks to ensure that the application is healthy and no more. Ensure that the following conditions are met for the application:
  - Any required database or other external storage connections are up and running.
  - The load is acceptable for the server on which the application runs.
  - Your site is not in maintenance mode.
  - Tests specific to your application are operational.
- The page generated by the health check should be small in size:
  - It returns in a sub-second interval.
  - It does not induce significant load on the application server.
- The page generated by the health check is never cached, although the code that runs the health check might reference cached data.
  For example, you might find it useful to run a more extensive health check using cron and store the results to disk. The code that generates the page at the health monitor **url-path** incorporates the results of this cron job in the tests it performs.

- Because the Load-balancing service only processes the HTTP status code returned, and because health checks are run so frequently, you can use the **HEAD** or **OPTIONS** HTTP methods to skip processing the entire page.

CHAPTER 8. CREATING NON-SECURE HTTP LOAD BALANCERS

You can create the following load balancers for non-secure HTTP network traffic:

- Section 8.1, “Creating an HTTP load balancer with a health monitor”
- Section 8.2, “Creating an HTTP load balancer that uses a floating IP”
- Section 8.3, “Creating an HTTP load balancer with session persistence”

8.1. CREATING AN HTTP LOAD BALANCER WITH A HEALTH MONITOR

For networks that are not compatible with Red Hat OpenStack Platform Networking service (neutron) floating IPs, create a load balancer to manage network traffic for non-secure HTTP applications. Create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on a public subnet (**public_subnet**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id public_subnet --wait
   ```

3. Create a listener (**listener1**) on a port (**80**).

   Example

   ```
   $ openstack loadbalancer listener create --name listener1 \
   --protocol HTTP --protocol-port 80 lb1
   ```

4. Verify the state of the listener.

   Example

```
$ openstack loadbalancer listener show listener1
```

Before going to the next step, ensure that the status is **ACTIVE**.

5. Create the listener default pool (**pool1**).

Example

```
$ openstack loadbalancer pool create --name pool1 \
--lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP
```

6. Create a health monitor (**healthmon1**) of type (**HTTP**) on the pool (**pool1**) that connects to the back-end servers and tests the path (/).
Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

Example

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
```

7. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the default pool.

Example

In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

```
$ openstack loadbalancer member create --name member1 --subnet-id \
private_subnet --address 192.0.2.10 --protocol-port 80 pool1

$ openstack loadbalancer member create --name member2 --subnet-id \
private_subnet --address 192.0.2.11 --protocol-port 80 pool1
```

Verification

1. View and verify the load balancer (lb1) settings:

Example

```
$ openstack loadbalancer show lb1
```

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| description         |                                      |
| flavor              |                                      |
| id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-01-15T11:12:13                  |
| vip_address         | 198.51.100.12                        |
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. When a health monitor is present and functioning properly, you can check the status of each member.
   A working member (**member1**) has an **ONLINE** value for its **operating_status**.

   ### Example

   ```
   $ openstack loadbalancer member show pool1 member1
   ```

   ### Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:16:23                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                | member1                              |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2022-01-15T11:20:45                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

Additional resources

- loadbalancer in the *Command line interface reference*

8.2. CREATING AN HTTP LOAD BALANCER THAT USES A FLOATING IP

To manage network traffic for non-secure HTTP applications, create a load balancer with a virtual IP (VIP) that depends on a floating IP. The advantage of using a floating IP is that you retain control of the assigned IP, which is necessary if you need to move, destroy, or recreate your load balancer. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

NOTE

Floating IPs do not work with IPv6 networks.

Prerequisites

- A floating IP to use with a load balancer VIP.
- A Red Hat OpenStack Platform Networking service (neutron) shared external (public) subnet that you can reach from the internet to use for the floating IP.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on a private subnet (**private_subnet**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id private_subnet --wait
   ```

3. In the output from step 2, record the value of **load_balancer_vip_port_id**, because you must provide it in a later step.

4. Create a listener (**listener1**) on a port (**80**).

   Example

   ```
   $ openstack loadbalancer listener create --name listener1 \
   --protocol HTTP --protocol-port 80 lb1
   ```

5. Create the listener default pool (**pool1**).

   Example

   The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host non-secure HTTP applications on TCP port 80:

   ```
   $ openstack loadbalancer pool create --name pool1 \
   --lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP
   ```

6. Create a health monitor (**healthmon1**) of type (**HTTP**) on the pool (**pool1**) that connects to the back-end servers and tests the path (/).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   ### Example

   ```
   $ openstack loadbalancer healthmonitor create --name healthmon1 \
   --delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
   ```

7. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet to the default pool.

   ### Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 80 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 80 pool1
   ```

8. Create a floating IP address on the shared external subnet (**public**).

   ### Example

   ```
   $ openstack floating ip create public
   ```

9. In the output from step 8, record the value of **floating_ip_address**, because you must provide it in a later step.

10. Associate this floating IP (**203.0.113.0**) with the load balancer **vip_port_id** (**69a85edd-5b1c-458f-96f2-b4552b15b8e6**).

    ### Example

    ```
    $ openstack floating ip set --port 69a85edd-5b1c-458f-96f2-b4552b15b8e6 203.0.113.0
    ```

Verification

1. Verify HTTP traffic flows across the load balancer by using the floating IP (**203.0.113.0**).

   ### Example

   ```
   $ curl -v http://203.0.113.0 --insecure
   ```

   ### Sample output

   ```
   * About to connect() to 203.0.113.0 port 80 (#0)
   *   Trying 203.0.113.0...
   * Connected to 203.0.113.0 (203.0.113.0) port 80 (#0)
   ```

```
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 203.0.113.0
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 30
<
* Connection #0 to host 203.0.113.0 left intact
```

2. When a health monitor is present and functioning properly, you can check the status of each member.
   A working member (**member1**) has an **ONLINE** value for its **operating_status**.

   Example

   ```
   $ openstack loadbalancer member show pool1 member1
   ```

   Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.02.10                          |
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:23                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                | member1                              |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2022-01-15T11:28:42                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

Additional resources

- loadbalancer in the *Command line interface reference*
- floating in the *Command line interface reference*

8.3. CREATING AN HTTP LOAD BALANCER WITH SESSION PERSISTENCE

To manage network traffic for non-secure HTTP applications, you can create load balancers that track session persistence. Doing so ensures that when a request comes in, the load balancer directs subsequent requests from the same client to the same back-end server. Session persistence optimizes load balancing by saving time and memory.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.
- The non-secure web applications whose network traffic you are load balancing have cookies enabled.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on a public subnet (**public_subnet**).

   > **NOTE**
   >
   > Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id public_subnet --wait
   ```

3. Create a listener (**listener1**) on a port (**80**).

   Example

   ```
   $ openstack loadbalancer listener create --name listener1 \
   --protocol HTTP --protocol-port 80 lb1
   ```

4. Create the listener default pool (**pool1**) that defines session persistence on a cookie (**PHPSESSIONID**).

   Example

   The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host non-secure HTTP applications on TCP port 80:

   ```
   $ openstack loadbalancer pool create --name pool1 \
   --lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP \
   --session-persistence type=APP_COOKIE,cookie_name=PHPSESSIONID
   ```

5. Create a health monitor (**healthmon1**) of type (**HTTP**) on the pool (**pool1**) that connects to the back-end servers and tests the path (/).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   Example

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
```

6. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the default pool.

   Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 80 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 80 pool1
   ```

Verification

1. View and verify the load balancer (**lb1**) settings:

   Example

   ```
   $ openstack loadbalancer show lb1
   ```

   Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:58                  |
   | description         |                                      |
   | flavor              |                                      |
   | id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
   | listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
   | name                | lb1                                  |
   | operating_status    | ONLINE                               |
   | pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | provider            | amphora                              |
   | provisioning_status | ACTIVE                               |
   | updated_at          | 2022-01-15T11:28:42                  |
   | vip_address         | 198.51.100.22                        |
   | vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
   | vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
   | vip_qos_policy_id   | None                                 |
   | vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   +---------------------+--------------------------------------+
   ```

2. When a health monitor is present and functioning properly, you can check the status of each member.
   A working member (**member1**) has an **ONLINE** value for its **operating_status**.

Example

```
$ openstack loadbalancer member show pool1 member1
```

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.02.10                          |
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:23                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                | member1                              |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 80                                   |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2022-01-15T11:28:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

Additional resources

- loadbalancer in the *Command line interface reference*

CHAPTER 9. CREATING SECURE HTTP LOAD BALANCERS

You can create various types of load balancers to manage secure HTTP (HTTPS) network traffic.

9.1. ABOUT NON-TERMINATED HTTPS LOAD BALANCERS

A non-terminated HTTPS load balancer acts effectively like a generic TCP load balancer: the load balancer forwards the raw TCP traffic from the web client to the back-end servers where the HTTPS connection is terminated with the web clients. While non-terminated HTTPS load balancers do not support advanced load balancer features like Layer 7 functionality, they do lower load balancer resource utilization by managing the certificates and keys themselves.

9.2. CREATING A NON-TERMINATED HTTPS LOAD BALANCER

If your application requires HTTPS traffic to terminate on the back-end member servers, typically called *HTTPS pass through*, you can use the HTTPS protocol for your load balancer listeners.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on a public subnet (**public_subnet**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

```
$ openstack loadbalancer create --name lb1 \
--vip-subnet-id public_subnet --wait
```

3. Create a listener (**listener1**) on a port (**443**).

Example

```
$ openstack loadbalancer listener create --name listener1 \
--protocol HTTPS --protocol-port 443 lb1
```

4. Create the listener default pool (**pool1**).

Example

The command in this example creates an HTTPS pool that uses a private subnet containing back-end servers that host HTTPS applications configured with a TLS-encrypted web application on TCP port 443:

```
$ openstack loadbalancer pool create --name pool1 \
--lb-algorithm ROUND_ROBIN --listener listener1 \
--protocol HTTPS
```

5. Create a health monitor (**healthmon1**) on the pool (**pool1**) of type (**TLS-HELLO**) that connects to the back-end servers and tests the path (/).
Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

Example

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 15 --max-retries 4 --timeout 10 --type TLS-HELLO \
--url-path / pool1
```

6. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the default pool.

Example

In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

```
$ openstack loadbalancer member create --name member1 --subnet-id \
private_subnet --address 192.0.2.10 --protocol-port 443 pool1

$ openstack loadbalancer member create --name member2 --subnet-id \
private_subnet --address 192.0.2.11 --protocol-port 443 pool1
```

Verification

1. View and verify the load balancer (**lb1**) settings.

Example

```
$ openstack loadbalancer show lb1
```

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| description         |                                      |
| flavor              |                                      |
| id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-01-15T11:12:42                  |
| vip_address         | 198.51.100.11                        |
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. When a health monitor is present and functioning properly, you can check the status of each member.

Example

A working member (**member1**) has an **ONLINE** value for its **operating_status**.

```
$ openstack loadbalancer member show pool1 member1
```

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                | member1                              |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 443                                  |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2022-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

Additional resources

- *Managing secrets with the Key Manager service* guide
- loadbalancer in the *Command line interface reference*

9.3. ABOUT TLS-TERMINATED HTTPS LOAD BALANCERS

When a TLS-terminated HTTPS load balancer is implemented, web clients communicate with the load balancer over Transport Layer Security (TLS) protocols. The load balancer terminates the TLS session and forwards the decrypted requests to the back-end servers. When you terminate the TLS session on the load balancer, you offload the CPU-intensive encryption operations to the load balancer, and allow the load balancer to use advanced features such as Layer 7 inspection.

9.4. CREATING A TLS-TERMINATED HTTPS LOAD BALANCER

When you use TLS-terminated HTTPS load balancers, you offload the CPU-intensive encryption operations to the load balancer, and allow the load balancer to use advanced features such as Layer 7 inspection. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.
- TLS public-key cryptography is configured with the following characteristics:
  - A TLS certificate, key, and intermediate certificate chain is obtained from an external certificate authority (CA) for the DNS name that is assigned to the load balancer VIP address, for example, **www.example.com**.
  - The certificate, key, and intermediate certificate chain reside in separate files in the current directory.
  - The key and certificate are PEM-encoded.
  - The intermediate certificate chain contains multiple certificates that are PEM-encoded and concatenated together.
- You must configure the Load-balancing service (octavia) to use the Key Manager service (barbican). For more information, see the *Managing secrets with the Key Manager service* guide.

Procedure

1. Combine the key (**server.key**), certificate (**server.crt**), and intermediate certificate chain (**ca-chain.crt**) into a single PKCS12 file (**server.p12**).

NOTE

Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

Example

```
$ openssl pkcs12 -export -inkey server.key -in server.crt \
-certfile ca-chain.crt -passout pass: -out server.p12
```

NOTE

The following procedure does not work if you password protect the PKCS12 file.

2. Source your credentials file.

   ### Example

   ```
   $ source ~/overcloudrc
   ```

3. Use the Key Manager service to create a secret resource (**tls_secret1**) for the PKCS12 file.

   ### Example

   ```
   $ openstack secret store --name='tls_secret1' \
   -t 'application/octet-stream' -e 'base64' \
   --payload="$(base64 < server.p12)"
   ```

4. Create a load balancer (**lb1**) on the public subnet (**public_subnet**).

   ### Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id public_subnet --wait
   ```

5. Create a **TERMINATED_HTTPS** listener (**listener1**), and reference the secret resource as the default TLS container for the listener.

   ### Example

   ```
   $ openstack loadbalancer listener create --protocol-port 443 \
   --protocol TERMINATED_HTTPS \
   --default-tls-container=\
   $(openstack secret list | awk '/ tls_secret1 / {print $2}') lb1
   ```

6. Create a pool (**pool1**) and make it the default pool for the listener.

   ### Example

   The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host non-secure HTTP applications on TCP port 80:

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener
listener1 --protocol HTTP
```

7. Create a health monitor (**healthmon1**) of type (**HTTP**) on the pool (**pool1**) that connects to the back-end servers and tests the path (/).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   ### Example

   ```
   $ openstack loadbalancer healthmonitor create --name healthmon1 \
   --delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
   ```

8. Add the non-secure HTTP back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool.

   ### Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 443 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 443 pool1
   ```

Verification

1. View and verify the load balancer (**lb1**) settings.

   ### Example

   ```
   $ openstack loadbalancer show lb1
   ```

   ### Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:09                  |
   | description         |                                      |
   | flavor              |                                      |
   | id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
   | listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
   | name                | lb1                                  |
   | operating_status    | ONLINE                               |
   | pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | provider            | amphora                              |
   | provisioning_status | ACTIVE                               |
   | updated_at          | 2022-01-15T11:12:42                  |
   | vip_address         | 198.51.100.11                        |
   ```

```
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. When a health monitor is present and functioning properly, you can check the status of each member.

Example

```
$ openstack loadbalancer member show pool1 member1
```

A working member (**member1**) has an **ONLINE** value for its **operating_status**:

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                | member1                              |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 80                                   |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2022-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

Additional resources

- *Managing secrets with the Key Manager service* guide
- loadbalancer in the *Command line interface reference*

9.5. CREATING A TLS-TERMINATED HTTPS LOAD BALANCER WITH SNI

For TLS-terminated HTTPS load balancers that employ Server Name Indication (SNI) technology, a single listener can contain multiple TLS certificates and enable the load balancer to know which certificate to present when it uses a shared IP. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.
- TLS public-key cryptography is configured with the following characteristics:
    - Multiple TLS certificates, keys, and intermediate certificate chains have been obtained from an external certificate authority (CA) for the DNS names assigned to the load balancer VIP address, for example, **www.example.com** and **www2.example.com**.
    - The keys and certificates are PEM-encoded.
- You must configure the Load-balancing service (octavia) to use the Key Manager service (barbican). For more information, see the *Managing secrets with the Key Manager service* guide.

Procedure

1. For each of the TLS certificates in the SNI list, combine the key (**server.key**), certificate (**server.crt**), and intermediate certificate chain (**ca-chain.crt**) into a single PKCS12 file (**server.p12**).
   In this example, you create two PKCS12 files (**server.p12** and **server2.p12**) one for each certificate (**www.example.com** and **www2.example.com**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   #### Example

   ```
   $ openssl pkcs12 -export -inkey server.key -in server.crt \
   -certfile ca-chain.crt -passout pass: -out server.p12

   $ openssl pkcs12 -export -inkey server2.key -in server2.crt \
   -certfile ca-chain2.crt -passout pass: -out server2.p12
   ```

2. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

3. Use the Key Manager service to create secret resources (**tls_secret1** and **tls_secret2**) for the PKCS12 file.

   #### Example

   ```
   $ openstack secret store --name='tls_secret1' \
   -t 'application/octet-stream' -e 'base64' \
   --payload="$(base64 < server.p12)"

   $ openstack secret store --name='tls_secret2' \
   -t 'application/octet-stream' -e 'base64' \
   --payload="$(base64 < server2.p12)"
   ```

4. Create a load balancer (**lb1**) on the public subnet (**public_subnet**).

   Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id public_subnet --wait
   ```

5. Create a TERMINATED_HTTPS listener (**listener1**), and use SNI to reference both the secret resources.
   (Reference **tls_secret1** as the default TLS container for the listener.)

   Example

   ```
   $ openstack loadbalancer listener create  --name listener1 \
   --protocol-port 443 --protocol TERMINATED_HTTPS \
   --default-tls-container=\
   $(openstack secret list | awk '/ tls_secret1 / {print $2}') \
   --sni-container-refs \
   $(openstack secret list | awk '/ tls_secret1 / {print $2}') \
   $(openstack secret list | awk '/ tls_secret2 / {print $2}') -- lb1
   ```

6. Create a pool (**pool1**) and make it the default pool for the listener.

   Example

   The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host non-secure HTTP applications on TCP port 80:

   ```
   $ openstack loadbalancer pool create --name pool1 \
   --lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP
   ```

7. Create a health monitor (**healthmon1**) of type (**HTTP**) on the pool (**pool1**) that connects to the back-end servers and tests the path (/).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   Example

   ```
   $ openstack loadbalancer healthmonitor create --name healthmon1 \
   --delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
   ```

8. Add the non-secure HTTP back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool.

   Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 443 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 443 pool1
   ```

Verification

1. View and verify the load balancer (**lb1**) settings.

   #### Example

   ```
   $ openstack loadbalancer show lb1
   ```

   #### Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:09                  |
   | description         |                                      |
   | flavor              |                                      |
   | id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
   | listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
   | name                | lb1                                  |
   | operating_status    | ONLINE                               |
   | pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | provider            | amphora                              |
   | provisioning_status | ACTIVE                               |
   | updated_at          | 2022-01-15T11:12:42                  |
   | vip_address         | 198.51.100.11                        |
   | vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
   | vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
   | vip_qos_policy_id   | None                                 |
   | vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   +---------------------+--------------------------------------+
   ```

2. When a health monitor is present and functioning properly, you can check the status of each member.

   #### Example

   ```
   $ openstack loadbalancer member show pool1 member1
   ```

   #### Sample output

   A working member (**member1**) has an **ONLINE** value for its **operating_status**:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:09                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                | member1                              |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   ```

```
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2022-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

Additional resources

- *Managing secrets with the Key Manager service* guide
- loadbalancer in the *Command line interface reference*

9.6. CREATING A TLS-TERMINATED LOAD BALANCER WITH AN HTTP/2 LISTENER

When you use TLS-terminated HTTPS load balancers, you offload the CPU-intensive encryption operations to the load balancer, and allow the load balancer to use advanced features such as Layer 7 inspection. With the addition of an HTTP/2 listener, you can leverage the HTTP/2 protocol to improve performance by loading pages faster. Load balancers negotiate HTTP/2 with clients by using the Application-Layer Protocol Negotiation (ALPN) TLS extension.

The Load-balancing service (octavia) supports end-to-end HTTP/2 traffic, which means that the HTTP2 traffic is not translated by HAProxy from the point where the request reaches the listener until the response returns from the load balancer. To achieve end-to-end HTTP/2 traffic, you must have an HTTP pool with back-end re-encryption: pool members that are listening on a secure port and web applications that are configured for HTTPS traffic.

You can send HTTP/2 traffic to an HTTP pool without back-end re-encryption. In this situation, HAProxy translates the traffic before it reaches the pool, and the response is translated back to HTTP/2 before it returns from the load balancer.

Red Hat recommends that you create a health monitor to ensure that your back-end members remain available.

NOTE

Currently, the Load-balancing service does not support health monitoring for TLS-terminated load balancers that use HTTP/2 listeners.

Prerequisites

- TLS public-key cryptography is configured with the following characteristics:
  - A TLS certificate, key, and intermediate certificate chain is obtained from an external certificate authority (CA) for the DNS name that is assigned to the load balancer VIP address, for example, **www.example.com**.
  - The certificate, key, and intermediate certificate chain reside in separate files in the current directory.
  - The key and certificate are PEM-encoded.

- The intermediate certificate chain contains multiple certificates that are PEM-encoded and concatenated together.

- You must configure the Load-balancing service (octavia) to use the Key Manager service (barbican). For more information, see the *Managing secrets with the Key Manager service* guide.

Procedure

1. Combine the key (**server.key**), certificate (**server.crt**), and intermediate certificate chain (**ca-chain.crt**) into a single PKCS12 file (**server.p12**).

   ### NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   ### IMPORTANT

   When you create the PKCS12 file, do not password protect the file.

   ### Example

   In this example, the PKCS12 file is created without a password:

   ```
   $ openssl pkcs12 -export -inkey server.key -in server.crt \
   -certfile ca-chain.crt -passout pass: -out server.p12
   ```

2. Source your credentials file.

   ### Example

   ```
   $ source ~/overcloudrc
   ```

3. Use the Key Manager service to create a secret resource (**tls_secret1**) for the PKCS12 file.

   ### Example

   ```
   $ openstack secret store --name='tls_secret1' \
   -t 'application/octet-stream' -e 'base64' \
   --payload="$(base64 < server.p12)"
   ```

4. Create a load balancer (**lb1**) on the public subnet (**public_subnet**).

   ### Example

   ```
   $ openstack loadbalancer create --name lb1 --vip-subnet-id \
   public_subnet --wait
   ```

5. Create a **TERMINATED_HTTPS** listener (**listener1**) and do the following:

   - reference the secret resource (**tls_secret1**) as the default TLS container for the listener.

- set the ALPN protocol (**h2**).
- set the fallback protocol if the client does not support HTTP/2 (**http/1.1**).

  ### Example

  ```
  $ openstack loadbalancer listener create --name listener1 \
  --protocol-port 443 --protocol TERMINATED_HTTPS --alpn-protocol h2 \
  --alpn-protocol http/1.1 --default-tls-container=\
  $(openstack secret list | awk '/ tls_secret1 / {print $2}') lb1
  ```

6. Create a pool (**pool1**) and make it the default pool for the listener.

   ### Example

   The command in this example creates an HTTP pool containing back-end servers that host HTTP applications configured with a web application on TCP port 80:

   ```
   $ openstack loadbalancer pool create --name pool1 \
   --lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP
   ```

7. Create a health monitor (**healthmon1**) of type (**TCP**) on the pool (**pool1**) that connects to the back-end servers.
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   ### Example

   ```
   $ openstack loadbalancer healthmonitor create --name healthmon1 \
   --delay 15  --max-retries 4 --timeout 10 --type TCP pool1
   ```

8. Add the HTTP back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool.

   ### Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 80 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 80 pool1
   ```

Verification

1. View and verify the load balancer (**lb1**) settings.

   ### Example

   ```
   $ openstack loadbalancer status show lb1
   ```

   ### Sample output

```
{
  "loadbalancer": {
    "id": "936dad29-4c3f-4f24-84a8-c0e6f10ed810",
    "name": "lb1",
    "operating_status": "ONLINE",
    "provisioning_status": "ACTIVE",
    "listeners": [
      {
        "id": "708b82c6-8a6b-4ec1-ae53-e619769821d4",
        "name": "listener1",
        "operating_status": "ONLINE",
        "provisioning_status": "ACTIVE",
        "pools": [
          {
            "id": "5ad7c678-23af-4422-8edb-ac3880bd888b",
            "name": "pool1",
            "provisioning_status": "ACTIVE",
            "operating_status": "ONLINE",
            "health_monitor": {
              "id": "4ad786ef-6661-4e31-a325-eca07b2b3dd1",
              "name": "healthmon1",
              "type": "TCP",
              "provisioning_status": "ACTIVE",
              "operating_status": "ONLINE"
            },
            "members": [
              {
                "id": "facca0d3-61a7-4b46-85e8-da6994883647",
                "name": "member1",
                "operating_status": "ONLINE",
                "provisioning_status": "ACTIVE",
                "address": "192.0.2.10",
                "protocol_port": 80
              },
              {
                "id": "2b0d9e0b-8e0c-48b8-aa57-90b2fde2eae2",
                "name": "member2",
                "operating_status": "ONLINE",
                "provisioning_status": "ACTIVE",
                "address": "192.0.2.11",
                "protocol_port": 80
              }
...
```

2. When a health monitor is present and functioning properly, you can check the status of each member.

Example

```
$ openstack loadbalancer member show pool1 member1
```

Sample output

A working member (**member1**) has an **ONLINE** value for its **operating_status**:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2023-11-16T20:08:01                  |
| id                  | facca0d3-61a7-4b46-85e8-da6994883647 |
| name                | member1                              |
| operating_status    | ONLINE                               |
| project_id          | 9b29c91f67314bd09eda9018616851cf     |
| protocol_port       | 80                                   |
| provisioning_status | ACTIVE                               |
| subnet_id           | 3b459c95-64d2-4cfa-b348-01aacc4b3fa9 |
| updated_at          | 2023-11-16T20:08:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
| tags                |                                      |
+---------------------+--------------------------------------+
```

Additional resources

- *Managing secrets with the Key Manager service* guide
- loadbalancer in the *Command line interface reference*

9.7. CREATING HTTP AND TLS-TERMINATED HTTPS LOAD BALANCING ON THE SAME IP AND BACK-END

You can configure a non-secure listener and a TLS-terminated HTTPS listener on the same load balancer and the same IP address when you want to respond to web clients with the exact same content, regardless if the client is connected with a secure or non-secure HTTP protocol. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.
- TLS public-key cryptography is configured with the following characteristics:
  - A TLS certificate, key, and optional intermediate certificate chain have been obtained from an external certificate authority (CA) for the DNS name assigned to the load balancer VIP address (for example, www.example.com).
  - The certificate, key, and intermediate certificate chain reside in separate files in the current directory.
  - The key and certificate are PEM-encoded.
  - The intermediate certificate chain contains multiple certificates that are PEM-encoded and concatenated together.
- You have configured the Load-balancing service (octavia) to use the Key Manager service (barbican). For more information, see the *Managing secrets with the Key Manager service* guide.

- The non-secure HTTP listener is configured with the same pool as the HTTPS TLS-terminated load balancer.

Procedure

1. Combine the key (**server.key**), certificate (**server.crt**), and intermediate certificate chain (**ca-chain.crt**) into a single PKCS12 file (**server.p12**).

   #### NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   #### Example

   ```
   $ openssl pkcs12 -export -inkey server.key -in server.crt \
   -certfile ca-chain.crt -passout pass: -out server.p12
   ```

2. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

3. Use the Key Manager service to create a secret resource (**tls_secret1**) for the PKCS12 file.

   #### Example

   ```
   $ openstack secret store --name='tls_secret1' \
   -t 'application/octet-stream' -e 'base64' \
   --payload="$(base64 < server.p12)"
   ```

4. Create a load balancer (**lb1**) on the public subnet (**public_subnet**).

   #### Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id external_subnet --wait
   ```

5. Create a TERMINATED_HTTPS listener (**listener1**), and reference the secret resource as the default TLS container for the listener.

   #### Example

   ```
   $ openstack loadbalancer listener create --name listener1 \
   --protocol-port 443 --protocol TERMINATED_HTTPS \
   --default-tls-container=\
   $(openstack secret list | awk '/ tls_secret1 / {print $2}') lb1
   ```

6. Create a pool (**pool1**) and make it the default pool for the listener.

Example

The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host non-secure HTTP applications on TCP port 80:

```
$ openstack loadbalancer pool create --name pool1 \
--lb-algorithm ROUND_ROBIN --listener listener1 --protocol HTTP
```

7. Create a health monitor (**healthmon1**) of type (**HTTP**) on the pool (**pool1**) that connects to the back-end servers and tests the path (/).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   ### Example

   ```
   $ openstack loadbalancer healthmonitor create --name healthmon1 \
   --delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
   ```

8. Add the non-secure HTTP back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool.

   ### Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 443 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 443 pool1
   ```

9. Create a non-secure, HTTP listener (**listener2**), and make its default pool, the same as the secure listener.

   ### Example

   ```
   $ openstack loadbalancer listener create --name listener2 \
   --protocol-port 80 --protocol HTTP --default-pool pool1 lb1
   ```

Verification

1. View and verify the load balancer (**lb1**) settings.

   ### Example

   ```
   $ openstack loadbalancer show lb1
   ```

   ### Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   ```

```
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| description         |                                      |
| flavor              |                                      |
| id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-01-15T11:12:42                  |
| vip_address         | 198.51.100.11                        |
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. When a health monitor is present and functioning properly, you can check the status of each member.

Example

```
$ openstack loadbalancer member show pool1 member1
```

Sample output

A working member (**member1**) has an **ONLINE** value for its **operating_status**:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                | member1                              |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 80                                   |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2022-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

Additional resources

- *Managing secrets with the Key Manager service* guide

- loadbalancer in the *Command line interface reference*

CHAPTER 10. CREATING OTHER KINDS OF LOAD BALANCERS

You use the Load-balancing service (octavia) to create the type of load balancer that matches the type of non-HTTP network traffic that you want to manage.

- Section 10.1, “Creating a TCP load balancer”
- Section 10.2, “Creating a UDP load balancer with a health monitor”
- Section 10.3, “Creating a QoS-ruled load balancer”
- Section 10.4, “Creating a load balancer with an access control list”
- Section 10.5, “Creating an OVN load balancer”

10.1. CREATING A TCP LOAD BALANCER

You can create a load balancer when you need to manage network traffic for non-HTTP, TCP-based services and applications. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on the public subnet (**public_subnet**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id public_subnet --wait
   ```

3. Create a **TCP** listener (**listener1**) on the specified port (**23456**) for which the custom application is configured.

   Example

   ```
   $ openstack loadbalancer listener create --name listener1 \
   --protocol TCP --protocol-port 23456 lb1
   ```

4. Create a pool (**pool1**) and make it the default pool for the listener.

   Example

   In this example, a pool is created that uses a private subnet containing back-end servers that host a custom application on a specific TCP port:

   ```
   $ openstack loadbalancer pool create --name pool1 \
   --lb-algorithm ROUND_ROBIN --listener listener1 \
   --protocol TCP
   ```

5. Create a health monitor (**healthmon1**) on the pool (**pool1**) that connects to the back-end servers and probes the TCP service port.

   Example

   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   ```
   $ openstack loadbalancer healthmonitor create --name healthmon1 \
   --delay 15 --max-retries 4 --timeout 10 --type TCP pool1
   ```

6. Add the back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool.

   Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 443 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 443 pool1
   ```

Verification

1. View and verify the load balancer (**lb1**) settings.

   Example

   ```
   $ openstack loadbalancer show lb1
   ```

   Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:09                  |
   | description         |                                      |
   | flavor              |                                      |
   | id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
   ```

```
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-01-15T11:12:42                  |
| vip_address         | 198.51.100.11                        |
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. When a health monitor is present and functioning properly, you can check the status of each member. Use the following command to obtain a member ID:

   Example

   ```
   $ openstack loadbalancer member list pool1
   ```

   A working member (**member1**) has an **ONLINE** value for its **operating_status**.

   Example

   ```
   $ openstack loadbalancer member show pool1 member1
   ```

   Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:09                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                | member1                              |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2022-01-15T11:12:42                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

Additional resources

- loadbalancer in the *Command line interface reference*

10.2. CREATING A UDP LOAD BALANCER WITH A HEALTH MONITOR

You can create a load balancer when you need to manage network traffic on UDP ports. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.
- No security rules that block ICMP Destination Unreachable messages (ICMP type 3).

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on a private subnet (**private_subnet**).

   > **NOTE**
   >
   > Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   #### Example

   ```
   $ openstack loadbalancer create --name lb1 \
   --vip-subnet-id private_subnet --wait
   ```

3. Create a listener (**listener1**) on a port (**1234**).

   #### Example

   ```
   $ openstack loadbalancer listener create --name listener1 \
   --protocol UDP --protocol-port 1234 lb1
   ```

4. Create the listener default pool (**pool1**).

   #### Example

   The command in this example creates a pool that uses a private subnet containing back-end servers that host one or more applications configured to use UDP ports:

   ```
   $ openstack loadbalancer pool create --name pool1 \
   --lb-algorithm ROUND_ROBIN --listener listener1 --protocol UDP
   ```

5. Create a health monitor (**healthmon1**) on the pool (**pool1**) that connects to the back-end servers by using UDP (**UDP-CONNECT**).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

Example

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 5 --max-retries 2 --timeout 3 --type UDP-CONNECT pool1
```

6. Add the back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the default pool.

Example

In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

```
$ openstack loadbalancer member create --name member1 --subnet-id \
private_subnet --address 192.0.2.10 --protocol-port 1234 pool1

$ openstack loadbalancer member create --name member2 --subnet-id \
private_subnet --address 192.0.2.11 --protocol-port 1234 pool1
```

Verification

1. View and verify the load balancer (**lb1**) settings.

Example

```
$ openstack loadbalancer show lb1
```

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| description         |                                      |
| flavor              |                                      |
| id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
| listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-01-15T11:12:42                  |
| vip_address         | 198.51.100.11                        |
| vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
| vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
+---------------------+--------------------------------------+
```

2. When a health monitor is present and functioning properly, you can check the status of each member.

Example

```
$ openstack loadbalancer member show pool1 member1
```

A working member (**member1**) has an **ONLINE** value for its **operating_status**.

Sample output

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2022-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                | member1                              |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 1234                                 |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2022-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

Additional resources

- loadbalancer in the *Command line interface reference*

10.3. CREATING A QOS-RULED LOAD BALANCER

You can apply a Red Hat OpenStack Platform (RHOSP) Networking service (neutron) Quality of Service (QoS) policy to virtual IP addresses (VIPs) that use load balancers. In this way, you can use a QoS policy to limit incoming or outgoing network traffic that the load balancer can manage. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.
- A QoS policy that contains bandwidth limit rules created for the RHOSP Networking service.

Procedure

1. Source your credentials file.

Example

```
$ source ~/overcloudrc
```

2. Create a network bandwidth QoS policy (**qos_policy_bandwidth**) with a maximum 1024 kbps and a maximum burst rate of 1024 kb.

> NOTE
>
> Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

Example

```
$ openstack network qos policy create qos_policy_bandwidth

$ openstack network qos rule create --type bandwidth-limit --max-kbps 1024 --max-burst-
kbits 1024 qos-policy-bandwidth
```

3. Create a load balancer (**lb1**) on the public subnet (**public_subnet**) by using a QoS policy (**qos-policy-bandwidth**).

Example

```
$ openstack loadbalancer create --name lb1 \
--vip-subnet-id public_subnet \
--vip-qos-policy-id qos-policy-bandwidth --wait
```

4. Create a listener (**listener1**) on a port (**80**).

Example

```
$ openstack loadbalancer listener create --name listener1 \
--protocol HTTP --protocol-port 80 lb1
```

5. Create the listener default pool (**pool1**).

Example

The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host an HTTP application on TCP port 80:

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener
listener1 --protocol HTTP
```

6. Create a health monitor (**healthmon1**) on the pool that connects to the back-end servers and tests the path (/).
Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

Example

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / \
pool1
```

7. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the default pool.

   ### Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 443 pool1

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 443 pool1
   ```

Verification

- View and verify the listener (**listener1**) settings.

  ### Example

  ```
  $ openstack loadbalancer list
  ```

  ### Sample output

  ```
  +---------------------+--------------------------------------+
  | Field               | Value                                |
  +---------------------+--------------------------------------+
  | admin_state_up      | True                                 |
  | created_at          | 2022-01-15T11:11:09                  |
  | description         |                                      |
  | flavor              |                                      |
  | id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
  | listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
  | name                | lb1                                  |
  | operating_status    | ONLINE                               |
  | pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
  | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
  | provider            | amphora                              |
  | provisioning_status | ACTIVE                               |
  | updated_at          | 2022-01-15T11:12:42                  |
  | vip_address         | 198.51.100.11                        |
  | vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
  | vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
  | vip_qos_policy_id   | cdfc3398-997b-46eb-9db1-ebbd88f7de05 |
  | vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
  +---------------------+--------------------------------------+
  ```

  In this example the parameter, **vip_qos_policy_id**, contains a policy ID.

Additional resources

- loadbalancer in the *Command line interface reference*

10.4. CREATING A LOAD BALANCER WITH AN ACCESS CONTROL LIST

You can create an access control list (ACL) to limit incoming traffic to a listener to a set of allowed source IP addresses. Any other incoming traffic is rejected. It is a best practice to also create a health monitor to ensure that your back-end members remain available.

Prerequisites

- A shared external (public) subnet that you can reach from the internet.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a load balancer (**lb1**) on the public subnet (**public_subnet**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer create --name lb1 --vip-subnet-id public_subnet --wait
   ```

3. Create a listener (**listener1**) with the allowed CIDRs (**192.0.2.0/24** and **198.51.100.0/24**).

   Example

   ```
   $ openstack loadbalancer listener create --name listener1 --protocol TCP --protocol-port 80 -
   -allowed-cidr 192.0.2.0/24 --allowed-cidr 198.51.100.0/24 lb1
   ```

4. Create the listener default pool (**pool1**).

   Example

   In this example, a pool is created that uses a private subnet containing back-end servers that are configured with a custom application on TCP port 80:

   ```
   $ openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --listener
   listener1 --protocol TCP
   ```

5. Create a health monitor on the pool that connects to the back-end servers and tests the path (**/**).
   Health checks are recommended but not required. If no health monitor is defined, the member server is assumed to be **ONLINE**.

   Example

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / pool1
```

6. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the default pool.

Example

In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

```
$ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.10 --
protocol-port 80 pool1

$ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.11 --
protocol-port 80 pool1
```

Verification

1. View and verify the listener (**listener1**) settings.

Example

```
$ openstack loadbalancer listener show listener1
```

Sample output

```
+-----------------------------+--------------------------------------+
| Field                       | Value                                |
+-----------------------------+--------------------------------------+
| admin_state_up              | True                                 |
| connection_limit            | -1                                   |
| created_at                  | 2022-01-15T11:11:09                  |
| default_pool_id             | None                                 |
| default_tls_container_ref   | None                                 |
| description                 |                                      |
| id                          | d26ba156-03c3-4051-86e8-f8997a202d8e |
| insert_headers              | None                                 |
| l7policies                  |                                      |
| loadbalancers               | 2281487a-54b9-4c2a-8d95-37262ec679d6 |
| name                        | listener1                            |
| operating_status            | ONLINE                               |
| project_id                  | 308ca9f600064f2a8b3be2d57227ef8f     |
| protocol                    | TCP                                  |
| protocol_port               | 80                                   |
| provisioning_status         | ACTIVE                               |
| sni_container_refs          | []                                   |
| timeout_client_data         | 50000                                |
| timeout_member_connect      | 5000                                 |
| timeout_member_data         | 50000                                |
| timeout_tcp_inspect         | 0                                    |
| updated_at                  | 2022-01-15T11:12:42                  |
| client_ca_tls_container_ref | None                                 |
| client_authentication       | NONE                                 |
| client_crl_container_ref    | None                                  |
| allowed_cidrs               | 192.0.2.0/24                          |
|                             | 198.51.100.0/24                       |
+-----------------------------+---------------------------------------+
```

In this example the parameter, **allowed_cidrs**, is set to allow traffic only from 192.0.2.0/24 and 198.51.100.0/24.

2. To verify that the load balancer is secure, ensure that a request to the listener from a client whose CIDR is not in the **allowed_cidrs** list; the request does not succeed.

Sample output

```
curl: (7) Failed to connect to 203.0.113.226 port 80: Connection timed out
curl: (7) Failed to connect to 203.0.113.226 port 80: Connection timed out
curl: (7) Failed to connect to 203.0.113.226 port 80: Connection timed out
curl: (7) Failed to connect to 203.0.113.226 port 80: Connection timed out
```

Additional resources

- loadbalancer in the *Command line interface reference*

10.5. CREATING AN OVN LOAD BALANCER

You can use the Red Hat OpenStack Platform (RHOSP) client to create a load balancer that manages network traffic in your RHOSP deployment. The RHOSP Load-Balancing service supports the neutron Modular Layer 2 plug-in with the Open Virtual Network mechanism driver (ML2/OVN).

Prerequisites

- The ML2/OVN provider driver must be deployed.

> **IMPORTANT**
>
> The OVN provider only supports Layer 4 TCP and UDP network traffic and the **SOURCE_IP_PORT** load balancer algorithm. The OVN provider does not support health monitoring.

- A shared external (public) subnet that you can reach from the internet.

Procedure

1. Source your credentials file.

Example

```
$ source ~/overcloudrc
```

2. Create a load balancer (**lb1**) on the private subnet (**private_subnet**) using the **--provider ovn** argument.

NOTE

Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

Example

```
$ openstack loadbalancer create --name lb1 --provider ovn \
--vip-subnet-id private_subnet --wait
```

3. Create a listener (**listener1**) that uses the protocol (**tcp**) on the specified port (**80**) for which the custom application is configured.

NOTE

The OVN provider only supports Layer 4 TCP and UDP network traffic.

Example

```
$ openstack loadbalancer listener create --name listener1 \
--protocol tcp --protocol-port 80 lb1
```

4. Create the listener default pool (**pool1**).

NOTE

The only supported load-balancing algorithm for OVN is **SOURCE_IP_PORT**.

Example

The command in this example creates an HTTP pool that uses a private subnet containing back-end servers that host a custom application on a specific TCP port:

```
$ openstack loadbalancer pool create --name pool1 --lb-algorithm \
SOURCE_IP_PORT --listener listener1 --protocol tcp
```

IMPORTANT

OVN does not support the health monitor feature for load-balancing.

5. Add the back-end servers (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool.

Example

In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

```
$ openstack loadbalancer member create --name member1 --subnet-id \
private_subnet --address 192.0.2.10 --protocol-port 80 pool1
```

```
$ openstack loadbalancer member create --name member2 --subnet-id \
private_subnet --address 192.0.2.11 --protocol-port 80 pool1
```

Verification

1. View and verify the load balancer (**lb1**) settings.

   #### Example

   ```
   $ openstack loadbalancer show lb1
   ```

   #### Sample output

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2022-01-15T11:11:09                  |
   | description         |                                      |
   | flavor              |                                      |
   | id                  | 788fe121-3dec-4e1b-8360-4020642238b0 |
   | listeners           | 09f28053-fde8-4c78-88b9-0f191d84120e |
   | name                | lb1                                  |
   | operating_status    | ONLINE                               |
   | pools               | 627842b3-eed8-4f5f-9f4a-01a738e64d6a |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | provider            | ovn                                  |
   | provisioning_status | ACTIVE                               |
   | updated_at          | 2022-01-15T11:12:42                  |
   | vip_address         | 198.51.100.11                        |
   | vip_network_id      | 9bca13be-f18d-49a5-a83d-9d487827fd16 |
   | vip_port_id         | 69a85edd-5b1c-458f-96f2-b4552b15b8e6 |
   | vip_qos_policy_id   | None                                 |
   | vip_subnet_id       | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   +---------------------+--------------------------------------+
   ```

2. Run the **openstack loadbalancer listener show** command to view the listener details.

   #### Example

   ```
   $ openstack loadbalancer listener show listener1
   ```

   #### Sample output

   ```
   +-----------------------------+--------------------------------------+
   | Field                       | Value                                |
   +-----------------------------+--------------------------------------+
   | admin_state_up              | True                                 |
   | connection_limit            | -1                                   |
   | created_at                  | 2022-01-15T11:13:52                  |
   | default_pool_id             | a5034e7a-7ddf-416f-9c42-866863def1f2 |
   | default_tls_container_ref   | None                                 |
   | description                 |                                      |
   ```

```
| id                          | a101caba-5573-4153-ade9-4ea63153b164 |
| insert_headers              | None                                 |
| l7policies                  |                                      |
| loadbalancers               | 653b8d79-e8a4-4ddc-81b4-e3e6b42a2fe3 |
| name                        | listener1                            |
| operating_status            | ONLINE                               |
| project_id                  | 7982a874623944d2a1b54fac9fe46f0b     |
| protocol                    | TCP                                  |
| protocol_port               | 64015                                |
| provisioning_status         | ACTIVE                               |
| sni_container_refs          | []                                   |
| timeout_client_data         | 50000                                |
| timeout_member_connect      | 5000                                 |
| timeout_member_data         | 50000                                |
| timeout_tcp_inspect         | 0                                    |
| updated_at                  | 2022-01-15T11:15:17                  |
| client_ca_tls_container_ref | None                                 |
| client_authentication       | NONE                                 |
| client_crl_container_ref    | None                                 |
| allowed_cidrs               | None                                 |
+-----------------------------+--------------------------------------+
```

3. Run the **openstack loadbalancer pool show** command to view the pool (**pool1**) and load-balancer members.

Example

```
$ openstack loadbalancer pool show pool1
```

Sample output

```
+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| admin_state_up       | True                                 |
| created_at           | 2022-01-15T11:17:34                  |
| description          |                                      |
| healthmonitor_id     |                                      |
| id                   | a5034e7a-7ddf-416f-9c42-866863def1f2 |
| lb_algorithm         | SOURCE_IP_PORT                       |
| listeners            | a101caba-5573-4153-ade9-4ea63153b164 |
| loadbalancers        | 653b8d79-e8a4-4ddc-81b4-e3e6b42a2fe3 |
| members              | 90d69170-2f73-4bfd-ad31-896191088f59 |
| name                 | pool1                                |
| operating_status     | ONLINE                               |
| project_id           | 7982a874623944d2a1b54fac9fe46f0b     |
| protocol             | TCP                                  |
| provisioning_status  | ACTIVE                               |
| session_persistence  | None                                 |
| updated_at           | 2022-01-15T11:18:59                  |
| tls_container_ref    | None                                 |
| ca_tls_container_ref | None                                 |
| crl_container_ref    | None                                 |
| tls_enabled          | False                                |
+----------------------+--------------------------------------+
```

Additional resources

- loadbalancer in the *Command line interface reference*

CHAPTER 11. IMPLEMENTING LAYER 7 LOAD BALANCING

You can use the Red Hat OpenStack Platform Load-balancing service (octavia) with layer 7 policies to redirect HTTP requests to particular application server pools by using several criteria to meet your business needs.

11.1. ABOUT LAYER 7 LOAD BALANCING

Layer 7 (L7) load balancing takes its name from the Open Systems Interconnection (OSI) model, indicating that the load balancer distributes requests to back end application server pools based on layer 7 (application) data. The following are different terms that all mean L7 load balancing: *request switching*, *application load balancing*, and *content-based- routing*, *switching*, or *balancing*. The Red Hat OpenStack Platform Load-balancing service (octavia) provides robust support for L7 load balancing.

NOTE

You cannot create L7 policies and rules with UDP load balancers.

An L7 load balancer consists of a listener that accepts requests on behalf of a number of back end pools and distributes those requests based on policies that use application data to determine which pools service any given request. This allows for the application infrastructure to be specifically tuned and optimized to serve specific types of content. For example, you can tune one group of back end servers (a pool) to serve only images; another for execution of server-side scripting languages like PHP and ASP; and another for static content such as HTML, CSS, and JavaScript.

Unlike lower-level load balancing, L7 load balancing does not require that all pools behind the load balancing service have the same content. L7 load balancers can direct requests based on URI, host, HTTP headers, and other data in the application message.

11.2. LAYER 7 LOAD BALANCING IN THE LOAD-BALANCING SERVICE

Although you can implement layer 7 (L7) load balancing for any well-defined L7 application interface, L7 functionality for the Red Hat OpenStack Platform Load-balancing service (octavia) refers only to the HTTP and TERMINATED_HTTPS protocols and its semantics.

Neutron LBaaS and the Load-balancing service use L7 rules and policies for the logic of L7 load balancing. An L7 rule is a single, simple logical test that evaluates to true or false. An L7 policy is a collection of L7 rules and a defined action to take if all the rules associated with the policy match.

11.3. LAYER 7 LOAD-BALANCING RULES

For the Red Hat OpenStack Platform Load-balancing service (octavia), a layer 7 (L7) load-balancing rule is a single, simple logical test that returns either true or false. It consists of a rule type, a comparison type, a value, and an optional key that is used depending on the rule type. An L7 rule must always be associated with an L7 policy.

NOTE

You cannot create L7 policies and rules with UDP load balancers.

Additional resources

- Section 11.4, “Layer 7 load-balancing rule types”

11.4. LAYER 7 LOAD-BALANCING RULE TYPES

The Red Hat OpenStack Platform Load-balancing service (octavia) has the following types of layer 7 load-balancing rules:

- **HOST_NAME**: The rule compares the HTTP/1.1 hostname in the request against the value parameter in the rule.

- **PATH**: The rule compares the path portion of the HTTP URI against the value parameter in the rule.

- **FILE_TYPE**: The rule compares the last portion of the URI against the value parameter in the rule, for example, txt, jpg, and so on.

- **HEADER**: The rule looks for a header defined in the key parameter and compares it against the value parameter in the rule.

- **COOKIE**: The rule looks for a cookie named by the key parameter and compares it against the value parameter in the rule.

- **SSL_CONN_HAS_CERT**: The rule matches if the client has presented a certificate for TLS client authentication. This does not imply that the certificate is valid.
- **SSL_VERIFY_RESULT**: This rule matches the TLS client authentication certificate validation result. A value of zero (**0**) means the certificate was successfully validated. A value greater than zero means the certificate failed validation. This value follows the **openssl-verify** result codes.
- **SSL_DN_FIELD**: The rule looks for a **Distinguished Name** field defined in the key parameter and compares it against the value parameter in the rule.

Additional resources

- Section 11.3, “Layer 7 load-balancing rules”

11.5. LAYER 7 LOAD-BALANCING RULE COMPARISON TYPES

For the Red Hat OpenStack Platform Load-balancing service (octavia), layer 7 load-balancing rules of a given type always perform comparisons. The Load-balancing service supports the following types of comparisons. Not all rule types support all comparison types:

- **REGEX**: Perl type regular expression matching
- **STARTS_WITH**: String starts with
- **ENDS_WITH**: String ends with
- **CONTAINS**: String contains
- **EQUAL_TO**: String is equal to

Additional resources

- Section 11.4, “Layer 7 load-balancing rule types”

11.6. LAYER 7 LOAD-BALANCING RULE RESULT INVERSION

To more fully express the logic that some policies require and the Red Hat OpenStack Platform Load-balancing service (octavia) uses, layer 7 load-balancing rules can have their result inverted. If the invert parameter of a given rule is true, the result of its comparison is inverted.

For example, an inverted **equal to** rule effectively becomes a **not equal to** rule. An inverted **regex** rule returns **true** only if the given regular expression does not match.

Additional resources

- Section 11.7, “Layer 7 load-balancing policies”

11.7. LAYER 7 LOAD-BALANCING POLICIES

For the Red Hat OpenStack Platform Load-balancing service (octavia), a layer 7 (L7) load-balancing policy is a collection of L7 rules associated with a listener, and which might also have an association to a back end pool. Policies are actions that the load balancer takes if all of the rules in the policy are true.

NOTE

You cannot create L7 policies and rules with UDP load balancers.

Additional resources

- Section 11.3, "Layer 7 load-balancing rules"

11.8. LAYER 7 LOAD-BALANCING POLICY LOGIC

The Red Hat OpenStack Platform Load-balancing service (octavia), layer 7 load-balancing policy uses the following logic: all the rules associated with a given policy are logically AND-ed together. A request must match all of the policy rules to match the policy.

If you need to express a logical OR operation between rules, create multiple policies with the same action or, make a more elaborate regular expression).

Additional resources

- Section 11.3, "Layer 7 load-balancing rules"

11.9. LAYER 7 LOAD-BALANCING POLICY ACTIONS

If the layer 7 load-balancing policy matches a given request, then that policy action is executed. The following are the actions an L7 policy might take:

- **REJECT**: The request is denied with an appropriate response code, and not forwarded on to any back end pool.
- **REDIRECT_TO_URL**: The request is sent an HTTP redirect to the URL defined in the redirect_url parameter.
- **REDIRECT_PREFIX**: Requests matching this policy are redirected to this prefix URL.
- **REDIRECT_TO_POOL**: The request is forwarded to the back-end pool associated with the L7 policy.

Additional resources

- Section 11.7, "Layer 7 load-balancing policies"

11.10. LAYER 7 LOAD-BALANCING POLICY POSITION

For the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia), when multiple layer 7 (L7) load-balancing policies are associated with a listener, then the value of the policy position parameter becomes important. The position parameter is used when determining the order that L7 policies are evaluated. The policy position affects listener behavior in the following ways:

- In the reference implementation of the Load-balancing service (haproxy amphorae), HAProxy enforces the following ordering regarding policy actions:
  - **REJECT** policies take precedence over all other policies.
  - **REDIRECT_TO_URL** policies take precedence over REDIRECT_TO_POOL policies.

  - **REDIRECT_TO_POOL** policies are evaluated only after all of the above, and in the order that the position of the policy specifies.

- L7 policies are evaluated in a specific order, as defined by the position attribute, and the first policy that matches a given request is the one whose action is followed.

- If no policy matches a given request, then the request is routed to the listener's default pool, if it exists. If the listener has no default pool, then an error 503 is returned.

- Policy position numbering starts with one (**1**).

- If a new policy is created with a position that matches that of an existing policy, then the new policy is inserted at the given position.

- If a new policy is created without specifying a position, or specifying a position that is greater than the number of policies already in the list, the new policy is appended to the list.

- When policies are inserted, deleted, or appended to the list, the policy position values are re-ordered from one (**1**) without skipping numbers. For example, if policy A, B, and C have position values of **1**, **2** and **3** respectively, if you delete policy B from the list, the position for policy C becomes **2**.

Additional resources

- Section 11.7, "Layer 7 load-balancing policies"

11.11. REDIRECTING UNSECURE HTTP REQUESTS TO SECURE HTTP

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) with layer 7 (L7) policies to redirect HTTP requests that are received on a non-secure TCP port to a secure TCP port.

In this example, any HTTP requests that arrive on the unsecure TCP port, 80, are redirected to the secure TCP port, 443.

Prerequisites

- A TLS-terminated HTTPS load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**). For more information, see Creating a TLS-terminated HTTPS load balancer .

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/overcloudrc
   ```

2. Create an HTTP listener (**http_listener**) on a load balancer (**lb1**) port (**80**).

   #### NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

Example

```
$ openstack loadbalancer listener create --name http_listener \
--protocol HTTP --protocol-port 80 lb1
```

3. Create an L7 policy (**policy1**) on the listener (**http_listener**). The policy must contain the action (**REDIRECT_TO_URL**) and point to the URL (**https://www.example.com/**).

   #### Example

   ```
   $ openstack loadbalancer l7policy create --name policy1 \
   --action REDIRECT_PREFIX --redirect-prefix https://www.example.com/ \
   http_listener
   ```

4. Add an L7 rule that matches all requests to a policy (**policy1**).

   #### Example

   ```
   $ openstack loadbalancer l7rule create --compare-type STARTS_WITH \
   --type PATH --value / policy1
   ```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policy, **policy1**, exists.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **STARTS_WITH** exists.

   #### Example

   ```
   $ openstack loadbalancer l7rule list policy1
   ```

Additional resources

- loadbalancer listener create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.12. REDIRECTING REQUESTS BASED ON THE STARTING PATH TO A POOL

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) to redirect HTTP requests to an alternate pool of servers. You can define a layer 7 (L7) policy to match one or more starting paths in the URL of the request.

In this example, any requests that contain URLs that begin with **/js** or **/images** are redirected to an alternate pool of static content servers.

Prerequisites

- An HTTP load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**). For more information, see Creating an HTTP load balancer with a health monitor .

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a second pool (**static_pool**) on a load balancer (**lb1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer pool create --name static_pool \
   --lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
   ```

3. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool (**static_pool**):

   Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 80 static_pool

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 80 static_pool
   ```

4. Create an L7 policy (**policy1**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the pool (**static_pool**).

   Example

   ```
   $ openstack loadbalancer l7policy create --name policy1 \
   --action REDIRECT_TO_POOL --redirect-pool static_pool listener1
   ```

5. Add an L7 rule that looks for **/js** at the start of the request path to the policy.

   Example

   ```
   $ openstack loadbalancer l7rule create --compare-type STARTS_WITH \
   --type PATH --value /js policy1
   ```

6. Create an L7 policy (**policy2**) with an action (**REDIRECT_TO_POOL**) and add the listener (**listener1**) pointed at the pool.

   Example

   ```
   $ openstack loadbalancer l7policy create --name policy2 \
   --action REDIRECT_TO_POOL --redirect-pool static_pool listener1
   ```

7. Add an L7 rule that looks for **/images** at the start of the request path to the policy.

   Example

   ```
   $ openstack loadbalancer l7rule create --compare-type STARTS_WITH \
   --type PATH --value /images policy2
   ```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policies, **policy1** and **policy2**, exist.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **STARTS_WITH** exists for each respective policy.

   Example

   ```
   $ openstack loadbalancer l7rule list policy1

   $ openstack loadbalancer l7rule list policy2
   ```

Additional resources

- loadbalancer pool create in the *Command line interface reference*
- loadbalancer member create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.13. SENDING SUBDOMAIN REQUESTS TO A SPECIFIC POOL

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) with layer 7 (L7) policies to redirect requests containing a specific HTTP/1.1 hostname to a different pool of application servers.

In this example, any requests that contain the HTTP/1.1 hostname, **www2.example.com**, are redirected to an alternate pool application servers, **pool2**.

Prerequisites

- An HTTP load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**).
  For more information, see Creating an HTTP load balancer with a health monitor .

Procedure

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a second pool (**pool2**) on the load balancer (**lb1**).

   > **NOTE**
   >
   > Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer pool create --name pool2 \
   --lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
   ```

3. Create an L7 policy (**policy1**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the pool (**pool2**).

   Example

   ```
   $ openstack loadbalancer l7policy create --name policy1 \
   --action REDIRECT_TO_POOL --redirect-pool pool2 listener1
   ```

4. Add an L7 rule to the policy that sends any requests using the HTTP/1.1 hostname, www2.example.com, to the second pool (**pool2**).

   Example

   ```
   $ openstack loadbalancer l7rule create --compare-type EQUAL_TO \
   --type HOST_NAME --value www2.example.com policy1
   ```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policy, **policy1**, exists.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **EQUAL_TO** exists for the policy.

   Example

   ```
   $ openstack loadbalancer l7rule list policy1
   ```

Additional resources

- loadbalancer pool create in the *Command line interface reference*

- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.14. SENDING REQUESTS BASED ON THE HOST NAME ENDING TO A SPECIFIC POOL

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) with layer 7 (L7) policies to redirect requests containing an HTTP/1.1 hostname that ends in a specific string to a different pool of application servers.

In this example, any requests that contain an HTTP/1.1 hostname that ends with, **.example.com**, are redirected to an alternate pool application server, **pool2**.

Prerequisites

- An HTTP load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**).
  For more information, see Creating an HTTP load balancer with a health monitor

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a second pool (**pool2**) on the load balancer (**lb1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer pool create --name pool2 \
   --lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
   ```

3. Create an L7 policy (**policy1**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the pool (**pool2**).

   Example

   ```
   $ openstack loadbalancer l7policy create --name policy1 \
   --action REDIRECT_TO_POOL --redirect-pool pool2 listener1
   ```

4. Add an L7 rule to the policy that sends any requests that use an HTTP/1.1 hostname (**www2.example.com**) to the second pool (**pool2**).

   Example

```
$ openstack loadbalancer l7rule create --compare-type ENDS_WITH \
--type HOST_NAME --value .example.com policy1
```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policy, **policy1**, exists.
2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **EQUAL_TO** exists for the policy.

   Example

   ```
   $ openstack loadbalancer l7rule list policy1
   ```

Additional resources

- loadbalancer pool create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.15. SENDING REQUESTS BASED ON ABSENCE OF A BROWSER COOKIE TO A SPECIFIC POOL

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) to redirect unauthenticated web client requests to a different pool that contains one or more authentication servers. A layer 7 (L7) policy determines whether the incoming request is missing an authentication cookie.

In this example, any web client requests that lack the browser cookie, **auth_token**, are redirected to an alternate pool that contains an authentication server.

> **IMPORTANT**
>
> This procedure provides an example for how to perform L7 application routing by using a browser cookie, and does not address security concerns.

Prerequisites

- A TLS-terminated HTTPS load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**). For more information, see Creating a TLS-terminated HTTPS load balancer.
- A second RHOSP Networking service (neutron) subnet on which a secure authentication server authenticates web users.

Procedure

1. Source your credentials file.

   Example

```
$ source ~/overcloudrc
```

2. Create a second pool (**login_pool**) on the load balancer (**lb1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   ### Example

   ```
   $ openstack loadbalancer pool create --name login_pool \
   --lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
   ```

3. Add a member, the secure authentication server (**192.0.2.10**) on the authenticating subnet (**secure_subnet**), to the second pool.

   ### Example

   In this example, the back-end server, **192.0.2.10**, is named **member1**:

   ```
   $ openstack loadbalancer member create --name member1 \
   --address 192.0.2.10 --protocol-port 80 --subnet-id secure_subnet \
   login_pool
   ```

4. Create an L7 policy (**policy1**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the second pool (**login_pool**).

   ### Example

   ```
   $ openstack loadbalancer l7policy create --name policy1 \
   --action REDIRECT_TO_POOL --redirect-pool login_pool listener1
   ```

5. Add an L7 rule to the policy (**policy1**) that searches for the browser cookie (**auth_token**) with any value, and matches if the cookie is NOT present.

   ### Example

   ```
   $ openstack loadbalancer l7rule create --compare-type REGEX \
   --key auth_token --type COOKIE --value '.*' --invert policy1
   ```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policy, **policy1**, exists.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **REGEX** exists.

   ### Example

```
$ openstack loadbalancer l7rule list policy1
```

Additional resources

- loadbalancer pool create in the *Command line interface reference*
- loadbalancer member create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.16. SENDING REQUESTS BASED ON ABSENCE OF A BROWSER COOKIE OR INVALID COOKIE VALUE TO A SPECIFIC POOL

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) to redirect unauthenticated web client requests to a different pool that contains one or more authentication servers. A layer 7 (L7) policy determines whether the incoming request is missing an authentication cookie or contains an authentication cookie with a particular value.

In this example, any web client requests that either lacks the browser cookie, **auth_token**, or has **auth_token** with a value of **INVALID**, are redirected to an alternate pool that contains an authentication server.

> **IMPORTANT**
>
> This procedure provides an example for how to perform L7 application routing using a browser cookie, and does not address security concerns.

Prerequisites

- A TLS-terminated HTTPS load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**). For more information, see Creating a TLS-terminated HTTPS load balancer.
- A second RHOSP Networking service (neutron) subnet on which a secure authentication server authenticates web users.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a second pool (**login_pool**) on the load balancer (**lb1**).

   > **NOTE**
   >
   > Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

Example

```
$ openstack loadbalancer pool create --name login_pool \
--lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
```

3. Add a member, the secure authentication server (**192.0.2.10**) on the authenticating subnet (**secure_subnet**), to the second pool.

   ### Example

   In this example, the back-end server, **192.0.2.10**, is named **member1**:

   ```
   $ openstack loadbalancer member create --name member1 \
   --address 192.0.2.10 --protocol-port 80 --subnet-id secure_subnet \
   login_pool
   ```

4. Create an L7 policy (**policy1**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the second pool (**login_pool**).

   ### Example

   ```
   $ openstack loadbalancer l7policy create --action REDIRECT_TO_POOL \
   --redirect-pool login_pool --name policy1 listener1
   ```

5. Add an L7 rule to the policy (**policy1**) that searches for the browser cookie (**auth_token**) with any value, and matches if the cookie is NOT present.

   ### Example

   ```
   $ openstack loadbalancer l7rule create --compare-type REGEX \
   --key auth_token --type COOKIE --value '.*' --invert policy1
   ```

6. Create a second L7 policy (**policy2**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the second pool (**login_pool**).

   ### Example

   ```
   $ openstack loadbalancer l7policy create --action REDIRECT_TO_POOL \
   --redirect-pool login_pool --name policy2 listener1
   ```

7. Add an L7 rule to the second policy (**policy2**) that searches for the browser cookie (**auth_token**) and matches if the cookie value equals the string **INVALID**.

   ### Example

   ```
   $ openstack loadbalancer l7rule create --compare-type EQUAL_TO \
   --key auth_token --type COOKIE --value INVALID policy2
   ```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policies, **policy1** and **policy2**, exist.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **REGEX** exists for **policy1** and a rule with a **compare_type** of **EQUAL_TO** exists for **policy2**.

   Example

   ```
   $ openstack loadbalancer l7rule list policy1

   $ openstack loadbalancer l7rule list policy2
   ```

Additional resources

- loadbalancer pool create in the *Command line interface reference*
- loadbalancer member create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.17. SENDING REQUESTS TO A POOL WHOSE NAME MATCHES THE HOSTNAME AND PATH

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) to redirect web client requests that match certain criteria to an alternate pool of application servers. The business logic criteria is performed through a layer 7 (L7) policy that attempts to match a predefined hostname and request path.

In this example, any web client requests that either match the hostname **api.example.com**, and have **/api** at the start of the request path are redirected to an alternate pool, **api_pool**.

Prerequisites

- An HTTP load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**). For more information, see Creating an HTTP load balancer with a health monitor .

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a second pool (**api_pool**) on the load balancer (**lb1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

```
$ openstack loadbalancer pool create --name api_pool \
--lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
```

3. Add load balancer members (**192.0.2.10** and **192.0.2.11**) on the private subnet (**private_subnet**) to the pool (**static_pool**):

   Example

   In this example, the back-end servers, **192.0.2.10** and **192.0.2.11**, are named **member1** and **member2**, respectively:

   ```
   $ openstack loadbalancer member create --name member1 --subnet-id \
   private_subnet --address 192.0.2.10 --protocol-port 80 static_pool

   $ openstack loadbalancer member create --name member2 --subnet-id \
   private_subnet --address 192.0.2.11 --protocol-port 80 static_pool
   ```

4. Create an L7 policy (**policy1**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the pool (**api_pool**).

   Example

   ```
   $ openstack loadbalancer l7policy create --action REDIRECT_TO_POOL \
   --redirect-pool api_pool --name policy1 listener1
   ```

5. Add an L7 rule to the policy that matches the hostname **api.example.com**.

   Example

   ```
   $ openstack loadbalancer l7rule create --compare-type EQUAL_TO \
   --type HOST_NAME --value api.example.com policy1
   ```

6. Add a second L7 rule to the policy that matches **/api** at the start of the request path. This rule is logically ANDed with the first rule.

   Example

   ```
   $ openstack loadbalancer l7rule create --compare-type STARTS_WITH \
   --type PATH --value /api policy1
   ```

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policy, **policy1**, exists.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that rules with a **compare_type** of **EQUAL_TO** and **STARTS_WITH**, respectively, both exist for **policy1**.

   Example

   ```
   $ openstack loadbalancer l7rule list policy1

   $ openstack loadbalancer l7rule list policy2
   ```

Additional resources

- loadbalancer pool create in the *Command line interface reference*
- loadbalancer member create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

11.18. CONFIGURING A-B TESTING ON AN EXISTING PRODUCTION SITE BY USING A COOKIE

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) with layer 7 (L7) policies to configure A-B testing, or split testing, for your production websites.

In this example, web clients that are routed to the “B” version of the website set the cookie **site_version** to **B** by the member servers in the pool (**pool1**).

Prerequisites

- Two production websites (site A and site B).
- You have configured an HTTP load balancer following the instructions for "Redirecting requests based on the starting path to a pool." A summary of the required configuration is:
  - Listener (**listener1**) on load balancer (**lb1**).
  - HTTP requests with a URL that starts with either /**js** or /**images** are sent to a pool (**static_pool**).
  - All other requests are sent to the listener default pool (**pool1**).
  - For more information about the configuration, see Section 11.12, “Redirecting requests based on the starting path to a pool”.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Create a third pool (**pool_B**) on a load balancer (**lb1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

```
$ openstack loadbalancer pool create --name pool_B \
--lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
```

3. Add load balancer members (**192.0.2.50** and **192.0.2.51**) on the private subnet (**private_subnet**) to the pool (**pool_B**):

   Example

   In this example, the back-end servers, **192.0.2.50** and **192.0.2.51**, are named **member1** and **member2**, respectively:

```
$ openstack loadbalancer member create --name member1 \
--address 192.0.2.50 --protocol-port 80 \
--subnet-id private_subnet pool_B

$ openstack loadbalancer member create --name member2 \
--address 192.0.2.51 --protocol-port 80 \
--subnet-id private_subnet pool_B
```

4. Create a fourth pool (**static_pool_B**) on a load balancer (**lb1**).

   Example

```
$ openstack loadbalancer pool create --name static_pool_B \
--lb-algorithm ROUND_ROBIN --loadbalancer lb1 --protocol HTTP
```

5. Add load balancer members (**192.0.2.100** and **192.0.2.101**) on the private subnet (**private_subnet**) to the pool (**static_pool_B**):

   Example

```
$ openstack loadbalancer member create --name member3 \
--address 192.0.2.100 --protocol-port 80 \
--subnet-id private_subnet static_pool_B

$ openstack loadbalancer member create --name member4 \
--address 192.0.2.101 --protocol-port 80 \
--subnet-id private_subnet static_pool_B
```

6. Create an L7 policy (**policy2**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the pool (**static_pool_B**). Insert the policy at position **1**.

   Example

```
$ openstack loadbalancer l7policy create --action REDIRECT_TO_POOL \
--redirect-pool static_pool_B --name policy2 --position 1 listener1
```

7. Add an L7 rule to the policy (**policy2**) that uses a regular expression to match either **/js** or **/images** at the start of the request path.

   Example

```
$ openstack loadbalancer l7rule create --compare-type REGEX \
--type PATH --value '^/(js|images)' policy2
```

8. Add a second L7 rule to the policy (**policy2**) that matches the cookie (**site_version**) to the exact string (**B**).

   Example

   ```
   $ openstack loadbalancer l7rule create --compare-type EQUAL_TO \
   --key site_version --type COOKIE --value B policy2
   ```

9. Create an L7 policy (**policy3**) on the listener (**listener1**). The policy must contain the action (**REDIRECT_TO_POOL**) and point to the pool (**pool_B**). Insert the policy at position **2**.

   Example

   ```
   $ openstack loadbalancer l7policy create --action REDIRECT_TO_POOL \
   --redirect-pool pool_B --name policy3 --position 2 listener1
   ```

10. Add an L7 rule to the policy (**policy3**) that matches the cookie (**site_version**) to the exact string (**B**).

    Example

    ```
    $ openstack loadbalancer l7rule create --compare-type EQUAL_TO \
    --key site_version --type COOKIE --value B policy3
    ```

> NOTE
>
> It is important to assign L7 policies with the most specific rules to a lower position, because the first policy whose rules all evaluate to True is the policy whose action is followed. In this procedure, **policy2** needs to be evaluated before **policy3** to avoid requests being sent to the incorrect pool.

Verification

1. Run the **openstack loadbalancer l7policy list** command and verify that the policies, **policy2** and **policy3**, exist.

2. Run the **openstack loadbalancer l7rule list <l7policy>** command and verify that a rule with a **compare_type** of **STARTS_WITH** exists for each respective policy.

   Example

   ```
   $ openstack loadbalancer l7rule list policy2

   $ openstack loadbalancer l7rule list policy3
   ```

Additional resources

- loadbalancer pool create in the *Command line interface reference*

- loadbalancer member create in the *Command line interface reference*
- loadbalancer l7policy create in the *Command line interface reference*
- loadbalancer l7rule create in the *Command line interface reference*

CHAPTER 12. GROUPING LOAD-BALANCING SERVICE OBJECTS BY USING TAGS

Tags are are arbitrary strings that you can add to Red Hat OpenStack Platform Load-balancing service (octavia) objects for the purpose of classifying them into groups. Tags do not affect the functionality of load-balancing objects: load balancers, listeners, pools, members, health monitors, rules, and polices. You can add a tag when you create the object, or add or remove a tag after the object has been created.

By associating a particular tag with load-balancing objects, you can run list commands to filter objects that belong to one or more groups. Being able to filter objects into one or more groups can be a starting point in managing usage, allocation, and maintenance of your load-balancing service resources. The ability to tag objects can also be leveraged by automated configuration management tools.

The topics included in this section are:

- Section 12.1, “Adding tags when creating Load-balancing service objects”
- Section 12.2, “Adding or removing tags on pre-existing Load-balancing service objects”
- Section 12.3, “Filtering Load-balancing service objects by using tags”

12.1. ADDING TAGS WHEN CREATING LOAD-BALANCING SERVICE OBJECTS

You can add a tag of your choice when you create a Red Hat OpenStack Load-balancing service (octavia) object. When the tags are in place, you can filter load balancers, listeners, pools, members, health monitors, rules, and policies by using their respective **loadbalancer list** commands.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Add a tag to a load-balancing object when you create it by using the **--tag <tag>** option with the appropriate **create** command for the object:

   - openstack loadbalancer create --tag <tag> ...
   - openstack loadbalancer listener create --tag <tag> ...
   - openstack loadbalancer pool create --tag <tag> ...
   - openstack loadbalancer member create --tag <tag> ...
   - openstack loadbalancer healthmonitor create --tag <tag> ...
   - openstack loadbalancer l7policy create --tag <tag> ...
   - openstack loadbalancer l7rule create --tag <tag> ...

NOTE

A tag can be any valid unicode string with a maximum length of 255 characters.

Example - creating and tagging a load balancer

In this example a load balancer, **lb1**, is created with two tags, **Finance** and **Sales**:

```
$ openstack loadbalancer create --name lb1 \
--vip-subnet-id public_subnet --tag Finance --tag Sales
```

NOTE

Load-balancing service objects can have one or more tags. Repeat the **--tag <tag>** option for each additional tag that you want to add.

Example - creating and tagging a listener

In this example a listener, **listener1**, is created with a tag, **Sales**:

```
$ openstack loadbalancer listener create --name listener1 --protocol HTTP --protocol-
port 80 --tag Sales lb1
```

Example - creating and tagging a pool

In this example a pool, **pool1**, is created with a tag, **Sales**:

```
$ openstack loadbalancer pool create --name pool1 \
--lb-algorithm ROUND_ROBIN --listener listener1 \
--protocol HTTP --tag Sales
```

Example - creating a member in a pool and tagging it

In this example a member, **192.0.2.10**, is created in **pool1** with a tag, **Sales**:

```
$ openstack loadbalancer member create --name member1 \
--subnet-id private_subnet --address 192.0.2.10 --protocol-port 80 \
--tag Sales pool1
```

Example - creating and tagging a health monitor

In this example a health monitor, **healthmon1**, is created with a tag, **Sales**:

```
$ openstack loadbalancer healthmonitor create --name healthmon1 \
--delay 15 --max-retries 4 --timeout 10 --type HTTP --url-path / \
--tag Sales pool1
```

Example - creating and tagging an L7 policy

In this example an L7 policy, **policy1**, is created with a tag, **Sales**:

```
$ openstack loadbalancer l7policy create --action REDIRECT_PREFIX \
--redirect-prefix https://www.example.com/ \
--name policy1 http_listener --tag Sales
```

Example - creating and tagging an L7 rule

In this example an L7 rule, **rule1**, is created with a tag, **Sales**:

```
$ openstack loadbalancer l7rule create --compare-type STARTS_WITH \
--type PATH --value / --tag Sales policy1
```

Verification

- Confirm that object that you created exists, and contains the tag that you added by using the appropriate **show** command for the object.

Example

In this example, the **show** command is run on the loadbalancer, **lb1**:

```
$ openstack loadbalancer show lb1
```

Sample output

```
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| availability_zone   | None                                 |
| created_at          | 2022-08-26T19:34:15                  |
| description         |                                      |
| flavor_id           | None                                 |
| id                  | 7975374b-3367-4436-ab19-2d79d8c1f29b |
| listeners           |                                      |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               |                                      |
| project_id          | 2eee3b86ca404cdd977281dac385fd4e     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-08-30T13:30:17                  |
| vip_address         | 172.24.3.76                          |
| vip_network_id      | 4c241fc4-95eb-491a-affe-26c53a8805cd |
| vip_port_id         | 9978a598-cc34-47f7-ba28-49431d570fd1 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | e999d323-bd0f-4469-974f-7f66d427e507 |
| tags                | Finance                              |
|                     | Sales                                |
+---------------------+--------------------------------------+
```

- loadbalancer create in the *Command line interface reference*
- loadbalancer show in the *Command line interface reference*

12.2. ADDING OR REMOVING TAGS ON PRE-EXISTING LOAD-BALANCING SERVICE OBJECTS

You can add and remove tags of your choice on Red Hat OpenStack Load-balancing service (octavia) objects after they have been created. When the tags are in place, you can filter load balancers, listeners, pools, members, health monitors, rules, and polices by using the their respective **loadbalancer list** commands.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Do one of the following:

   - Add a tag to a pre-existing load-balancing object by using the **--tag <tag>** option with the appropriate **set** command for the object:

     - **openstack loadbalancer set --tag <tag> <load_balancer_name_or_ID>**
     - **openstack loadbalancer listener set --tag <tag> <listener_name_or_ID>**
     - **openstack loadbalancer pool set --tag <tag> <pool_name_or_ID>**
     - **openstack loadbalancer member set --tag <tag> <pool_name_or_ID> <member_name_or_ID>**
     - **openstack loadbalancer healthmonitor set --tag <tag> <healthmon_name_or_ID>**
     - **openstack loadbalancer l7policy set --tag <tag> <l7policy_name_or_ID>**
     - **openstack loadbalancer l7rule set --tag <tag> <l7policy_name_or_ID> <l7rule_ID>**

     NOTE

     A tag can be any valid unicode string with a maximum length of 255 characters.

     Example

     In this example the tags, **Finance** and **Sales**, are added to the load balancer, **lb1**:

     ```
     $ openstack loadbalancer set --tag Finance --tag Sales lb1
     ```

     NOTE

     Load-balancing service objects can have one or more tags. Repeat the **--tag <tag>** option for each additional tag that you want to add.

   - Remove a tag from a pre-existing load-balancing object by using the **--tag <tag>** option with the appropriate **unset** command for the object:

  - **openstack loadbalancer unset --tag <tag> <load_balancer_name_or_ID>**
  - **openstack loadbalancer listener unset --tag <tag> <listener_name_or_ID>**
  - **openstack loadbalancer pool unset --tag <tag> <pool_name_or_ID>**
  - **openstack loadbalancer member unset --tag <tag> <pool_name_or_ID> <member_name_or_ID>**
  - **openstack loadbalancer healthmonitor unset --tag <tag> <healthmon_name_or_ID>**
  - **openstack loadbalancer l7policy unset --tag <tag> <policy_name_or_ID>**
  - **openstack loadbalancer l7rule unset --tag <tag> <policy_name_or_ID> <l7rule_ID>**

    Example

    In this example, the tag, **Sales**, is removed from the load balancer, **lb1**:

    ```
    $ openstack loadbalancer unset --tag Sales lb1
    ```

- Remove all tags from a pre-existing load-balancing object by using the **--no-tag** option with the appropriate **set** command for the object:
  - **openstack loadbalancer set --no-tag <load_balancer_name_or_ID>**
  - **openstack loadbalancer listener set --no-tag <listener_name_or_ID>**
  - **openstack loadbalancer pool set --no-tag <pool_name_or_ID>**
  - **openstack loadbalancer member set --no-tag <pool_name_or_ID> <member_name_or_ID>**
  - **openstack loadbalancer healthmonitor set --no-tag <healthmon_name_or_ID>**
  - **openstack loadbalancer l7policy set --no-tag <l7policy_name_or_ID>**
  - **openstack loadbalancer l7rule set --no-tag <l7policy_name_or_ID> <l7rule_ID>**

    Example

    In this example, all tags are removed from the load balancer, **lb1**:

    ```
    $ openstack loadbalancer set --no-tag lb1
    ```

Verification

- Confirm that you have added or removed one or more tags on the load-balancing object, by using the appropriate **show** command for the object.

  Example

  In this example, the **show** command is run on the loadbalancer, **lb1**:

  ```
  $ openstack loadbalancer show lb1
  ```

Sample output

```
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| availability_zone   | None                                 |
| created_at          | 2022-08-26T19:34:15                  |
| description         |                                      |
| flavor_id           | None                                 |
| id                  | 7975374b-3367-4436-ab19-2d79d8c1f29b |
| listeners           |                                      |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               |                                      |
| project_id          | 2eee3b86ca404cdd977281dac385fd4e     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-08-30T13:30:17                  |
| vip_address         | 172.24.3.76                          |
| vip_network_id      | 4c241fc4-95eb-491a-affe-26c53a8805cd |
| vip_port_id         | 9978a598-cc34-47f7-ba28-49431d570fd1 |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | e999d323-bd0f-4469-974f-7f66d427e507 |
| tags                | Finance                              |
|                     | Sales                                |
+---------------------+--------------------------------------+
```

- loadbalancer set in the *Command line interface reference*
- loadbalancer show in the *Command line interface reference*

12.3. FILTERING LOAD-BALANCING SERVICE OBJECTS BY USING TAGS

You can use the Red Hat OpenStack Load-balancing service (octavia) to create lists of objects. For the objects that are tagged, you can create filtered lists: lists that include or exclude objects based on whether your objects contain one or more of the specified tags. Being able to filter load balancers, listeners, pools, members, health monitors, rules, and policies using tags can be a starting point in managing usage, allocation, and maintenance of your load-balancing service resources.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Filter the objects that you want to list by running the appropriate **loadbalancer list** command for the objects with one of the tag options:

   Table 12.1. Tag options for filtering objects

In my list, I want to…	Examples
include objects that match all specified tags.	**$ openstack loadbalancer list --tags Sales,Finance** **$ openstack loadbalancer listener list --tags Sales,Finance** **$ openstack loadbalancer l7pool list --tags Sales,Finance** **$ openstack loadbalancer member list --tags Sales,Finance pool1** **$ openstack loadbalancer healthmonitor list --tags Sales,Finance** **$ openstack loadbalancer l7policy list --tags Sales,Finance** **$ openstack loadbalancer l7rule list --tags Sales,Finance policy1**
include objects that match one or more specified tags.	**$ openstack loadbalancer list --any-tags Sales,Finance** **$ openstack loadbalancer listener list --any-tags Sales,Finance** **$ openstack loadbalancer l7pool list --any-tags Sales,Finance** **$ openstack loadbalancer member list --any-tags Sales,Finance pool1** **$ openstack loadbalancer healthmonitor list --any-tags Sales,Finance** **$ openstack loadbalancer l7policy list --any-tags Sales,Finance** **$ openstack loadbalancer l7rule list --any-tags Sales,Finance policy1**

In my list, I want to...	Examples
exclude objects that match all specified tags.	**$ openstack loadbalancer list --not-tags Sales,Finance** **$ openstack loadbalancer listener list --not-tags Sales,Finance** **$ openstack loadbalancer l7pool list --not-tags Sales,Finance** **$ openstack loadbalancer member list --not-tags Sales,Finance pool1** **$ openstack loadbalancer healthmonitor list --not-tags Sales,Finance** **$ openstack loadbalancer l7policy list --not-tags Sales,Finance** **$ openstack loadbalancer l7rule list --not-tags Sales,Finance policy1**
exclude objects that match one or more specified tags.	**$ openstack loadbalancer list --not-any-tags Sales,Finance** **$ openstack loadbalancer listener list --not-any-tags Sales,Finance** **$ openstack loadbalancer l7pool list --not-any-tags Sales,Finance** **$ openstack loadbalancer member list --not-any-tags Sales,Finance pool1** **$ openstack loadbalancer healthmonitor list --not-any-tags Sales,Finance** **$ openstack loadbalancer l7policy list --not-any-tags Sales,Finance** **$ openstack loadbalancer l7rule list --not-any-tags Sales,Finance policy1**

NOTE

When specifying more than one tag, separate the tags by using a comma.

Additional resources

- loadbalancer list in the *Command line interface reference*

CHAPTER 13. CREATING AVAILABILITY ZONES FOR LOAD BALANCING NETWORK TRAFFIC AT THE EDGE

You can create load balancers in availability zones to increase traffic throughput and reduce latency by using the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia).

The topics included in this section are:

- Creating Load-balancing service availability zones
- Creating load balancers in availability zones

13.1. CREATING AVAILABILITY ZONES FOR THE LOAD-BALANCING SERVICE

With the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia), RHOSP administrators can create availability zones (AZs) that enable project users to create load balancers in a distributed compute node (DCN) environment to increase traffic throughput and reduce latency.

There are two steps required to create a Load-balancing service AZ: RHOSP administrators must first create an AZ profile, and then use the profile to create a Load-balancing service AZ that is visible to users.

An AZ profile must have the following:

- The name of the Compute service (nova) AZ.
- The management network to use.
  There are multiple management networks, one unique network for each AZ. The central AZ uses the existing load-balancing management network, **lb-mgmt-net**, and the additional AZs use their respective network, **lb-mgmt-<AZ_name>-net**, for example, **lb-mgmt-az-dcn1-net**, **lb-mgmt-az-dcn2-net**, and so on.

Prerequisites

- You must have a DCN environment in which the required networking resources have been created by running the **octavia-dcn-deployment.yaml** Ansible playbook.
  For more information, see Creating network resources for Load-balancing service availability zones in the *Deploying a Distributed Compute Node (DCN) architecture* guide.
- Your Load-balancing service provider driver must be amphora. The OVN provider driver does not support AZs.
- You must be a RHOSP user with the **admin** role.

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/centralrc
   ```

2. Gather the names of the Compute service AZs that you will use to name your Load-balancing service AZs.

   IMPORTANT

   The names of the Load-balancing service AZ that you create must match the names of your Compute service AZs.

   ```
   $ openstack availability zone list --compute
   ```

   ### Sample output

   ```
   +------------+-------------+
   | Zone Name  | Zone Status |
   +------------+-------------+
   | az-central | available   |
   | az-dcn1    | available   |
   | az-dcn2    | available   |
   | internal   | available   |
   +------------+-------------+
   ```

3. Gather the IDs for the management networks that you will use to create your Load-balancing service AZs:

   ```
   $ openstack network list -c Name -c ID
   ```

   ### Sample output

   ```
   +--------------------------------------+-------------------------+
   | ID                                   | Name                    |
   +--------------------------------------+-------------------------+
   | 10458d6b-e7c9-436f-92d9-711677c9d9fd | lb-mgmt-az-dcn2-net     |
   | 662a94f5-51eb-4a4c-86c4-52dcbf471ef9 | lb-mgmt-net             |
   | 6b97ef58-2a25-4ea5-931f-b7c07cd09474 | lb-mgmt-backbone-net    |
   | 99f4215b-fad8-432d-8444-1f894154dc30 | heat_tempestconf_network |
   | a2884aaf-846c-4936-9982-3083f6a71d9b | lb-mgmt-az-dcn1-net     |
   | d7f7de6c-0e84-49e2-9042-697fa85d2532 | public                  |
   | e887a9f9-15f7-4854-a797-033cedbfe5f3 | public2                 |
   +--------------------------------------+-------------------------+
   ```

4. Create an AZ profile. Repeat this step to create an AZ profile for each Load-balancing service AZ that you want to create:

   ```
   $ openstack loadbalancer availabilityzoneprofile create \
   --name <AZ_profile_name> --provider amphora --availability-zone-data '{"compute_zone": "
   <compute_AZ_name>","management_network": "<lb_mgmt_AZ_net_UUID>"}'
   ```

   ### Example - create profile for **az-central**

   In this example, an AZ profile (**az_profile_central**) is created that uses the management network (**lb-mgmt-net**) on a Compute node that runs in the Compute AZ (**az-central**):

   ```
   $ openstack loadbalancer availabilityzoneprofile create \
   ```

```
--name az_profile_central --provider amphora --availability-zone-data \
'{"compute_zone": "az-central","management_network": \
"662a94f5-51eb-4a4c-86c4-52dcbf471ef9"}'
```

5. Repeat step 4 to create an AZ profile for each Load-balancing service AZ that you want to create.

   ### Example - create profile for **az-dcn1**

   In this example, an AZ profile (**az-profile-dcn1**) is created that uses the management network (**lb-mgmt-az-dcn1-net**) on a Compute node that runs in the Compute AZ (**az-dcn1**):

   ```
   $ openstack loadbalancer availabilityzoneprofile create \
   --name az-profile-dcn1 --provider amphora --availability-zone-data \
   '{"compute-zone": "az-dcn1","management-network": \
   "a2884aaf-846c-4936-9982-3083f6a71d9b"}'
   ```

   ### Example - create profile for **az-dcn2**

   In this example, an AZ profile (**az-profile-dcn2**) is created that uses the management network (**lb-mgmt-az-dcn2-net**) on a Compute node that runs in the Compute AZ (**az-dcn2**):

   ```
   $ openstack loadbalancer availabilityzoneprofile create \
   --name az-profile-dcn2 --provider amphora --availability-zone-data \
   '{"compute-zone": "az-dcn2","management-network": \
   "10458d6b-e7c9-436f-92d9-711677c9d9fd"}'
   ```

6. Using the AZ profile, create a Load-balancing service AZ. Repeat this step for any additional AZs, using the appropriate profile for each AZ.

   ### Example - create AZ: az-central

   In this example, a Load-balancing service AZ (**az-central**) is created by using the AZ profile (**az-profile-central**):

   ```
   $ openstack loadbalancer availabilityzone create --name az-central \
   --availabilityzoneprofile az-profile-central \
   --description "AZ for Headquarters" --enable
   ```

   ### Example - create AZ: az-dcn1

   In this example, a Load-balancing service AZ (**az-dcn1**) is created by using the AZ profile (**az-profile-az-dcn1**):

   ```
   $ openstack loadbalancer availabilityzone create --name az-dcn1 \
   --availabilityzoneprofile az-profile-az-dcn1 \
   --description "AZ for South Region" --enable
   ```

   ### Example - create AZ: az-dcn2

   In this example, a Load-balancing service AZ (**az-dcn2**) is created by using the AZ profile (**az-profile-az-dcn2**):

```
$ openstack loadbalancer availabilityzone create --name az-dcn2 \
--availabilityzoneprofile az-profile-az-dcn2 \
--description "AZ for North Region" --enable
```

Verification

- Confirm that the AZ (**az-central**) was created. Repeat this step for any additional AZs, using the appropriate name for each AZ.

  Example - verify az-central

  ```
  $ openstack loadbalancer availabilityzone show az-central
  ```

  Sample output

  ```
  +------------------------------+--------------------------------------+
  | Field                        | Value                                |
  +------------------------------+--------------------------------------+
  | name                         | az-central                           |
  | availability_zone_profile_id | 5ed25d22-52a5-48ad-85ec-255910791623 |
  | enabled                      | True                                 |
  | description                  | AZ for Headquarters                  |
  +------------------------------+--------------------------------------+
  ```

  Example - verify az-dcn1

  ```
  $ openstack loadbalancer availabilityzone show az-dcn1
  ```

  Sample output

  ```
  +------------------------------+--------------------------------------+
  | Field                        | Value                                |
  +------------------------------+--------------------------------------+
  | name                         | az-dcn1                              |
  | availability_zone_profile_id | e0995a82-8e67-4cea-b32c-256cd61f9cf3 |
  | enabled                      | True                                 |
  | description                  | AZ for South Region                  |
  +------------------------------+--------------------------------------+
  ```

  Example - verify az-dcn2

  ```
  $ openstack loadbalancer availabilityzone show az-dcn2
  ```

  Sample output

  ```
  +------------------------------+--------------------------------------+
  | Field                        | Value                                |
  +------------------------------+--------------------------------------+
  | name                         | az-dcn2                              |
  | availability_zone_profile_id | 306a4725-7dac-4046-8f16-f2e668ee5a8d |
  ```

```
| enabled             | True                                 |
| description         | AZ for North Region                  |
+---------------------+--------------------------------------+
```

Next steps

- Creating load balancers in availability zones

Additional resources

- Creating network resources for Load-balancing service availability zones in the *Deploying a Distributed Compute Node (DCN) architecture* guide.
- loadbalancer availabilityzoneprofile create in the *Command line interface reference*
- loadbalancer availabilityzone create in the *Command line interface reference*

13.2. CREATING LOAD BALANCERS IN AVAILABILITY ZONES

With the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia), you can create load balancers in availability zones (AZs) in a distributed compute node (DCN) environment to increase traffic throughput and reduce latency.

Prerequisites

- You must have a Load-balancing service AZ provided by your RHOSP administrator.
- The virtual IP (VIP) network associated with the load balancer must be available in the AZ in which your load balancer is a member.

Procedure

1. Source your credentials file.

   #### Example

   ```
   $ source ~/centralrc
   ```

2. To create a load balancer for a DCN environment, use the **loadbalancer create** command with the **--availability-zone** option and specify the appropriate AZ.

   #### Example

   For example, to create a non-terminated HTTPS load balancer (**lb1**) on a public subnet (**public_subnet**) on availability zone (**az-central**), you would enter the following command:

   ```
   $ openstack loadbalancer create --name lb1 --vip-subnet-id \
   public_subnet --availability-zone az-central
   ```

3. Continue to create your load balancer by adding a listener, pool, health monitor, and load balancer members.

Verification

- Confirm that the load balancer (lb1) is a member of the availability zone (**az-central**).

  Example

  ```
  $ openstack loadbalancer show lb1
  ```

  Sample output

  ```
  +---------------------+--------------------------------------+
  | Field               | Value                                |
  +---------------------+--------------------------------------+
  | admin_state_up      | True                                 |
  | availability_zone   | az-central                           |
  | created_at          | 2023-07-12T16:35:05                  |
  | description         |                                      |
  | flavor_id           | None                                 |
  | id                  | 85c7e567-a0a7-4fcb-af89-a0bbc9abe3aa |
  | listeners           |                                      |
  | name                | lb1                                  |
  | operating_status    | ONLINE                               |
  | pools               |                                      |
  | project_id          | d303d3bda9b34d73926dc46f4d0cb4bc     |
  | provider            | amphora                              |
  | provisioning_status | ACTIVE                               |
  | updated_at          | 2023-07-12T16:36:45                  |
  | vip_address         | 10.101.10.229                        |
  | vip_network_id      | d7f7de6c-0e84-49e2-9042-697fa85d2532 |
  | vip_port_id         | 7f916764-d171-4317-9c86-a1750a54b16e |
  | vip_qos_policy_id   | None                                 |
  | vip_subnet_id       | a421cbcf-c5db-4323-b7ab-1df20ee6acab |
  | tags                |                                      |
  +---------------------+--------------------------------------+
  ```

Additional resources

- Creating availability zones for the Load-balancing service
- loadbalancer in the *Command line interface reference*

CHAPTER 14. UPDATING AND UPGRADING THE LOAD-BALANCING SERVICE

Perform regular updates and upgrades so that you can get the latest Red Hat OpenStack Platform Load-balancing service (octavia) features and bug fixes.

- Section 14.1, “Updating and upgrading the Load-balancing service”
- Section 14.2, “Updating running Load-balancing service instances”

14.1. UPDATING AND UPGRADING THE LOAD-BALANCING SERVICE

The Load-balancing service (octavia) is a part of a Red Hat OpenStack Platform (RHOSP) update or upgrade.

Prerequisites

- Schedule a maintenance window to perform the upgrade, because during the upgrade the Load-balancing service control plane is not fully functional.

Procedure

1. Perform the RHOSP update as described in the *Performing a minor update of Red Hat OpenStack Platform* guide.
2. After the maintenance release is applied, if you need to use new features, then rotate running amphorae to update them to the latest amphora image.

Additional resources

- *Performing a minor update of Red Hat OpenStack Platform* guide
- Section 14.2, “Updating running Load-balancing service instances”
- Section 2.2, “Load-balancing service (octavia) feature support matrix”

14.2. UPDATING RUNNING LOAD-BALANCING SERVICE INSTANCES

Periodically, you can update a running Load-balancing service instance (amphora) with a newer image. For example, you might want to update an amphora instance during the following events:

- An update or upgrade of Red Hat OpenStack Platform (RHOSP).
- A security update to your system.
- A change to a different flavor for the underlying virtual machine.

During an RHOSP update or upgrade, director automatically downloads the default amphora image, uploads it to the overcloud Image service (glance), and then configures the Load-balancing service (octavia) to use the new image. When you fail over a load balancer, you force the Load-balancing service to start an instance (amphora) that uses the new amphora image.

Prerequisites

- New images for amphora. These are available during an RHOSP update or upgrade.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. List the IDs for all of the load balancers that you want to update:

   ```
   $ openstack loadbalancer list -c id -f value
   ```

3. Fail over each load balancer:

   ```
   $ openstack loadbalancer failover <loadbalancer_id>
   ```

   NOTE

   When you start failing over the load balancers, monitor system utilization, and as needed, adjust the rate at which you perform failovers. A load balancer failover creates new virtual machines and ports, which might temporarily increase the load on OpenStack Networking.

4. Monitor the state of the failed over load balancer:

   ```
   $ openstack loadbalancer show <loadbalancer_id>
   ```

   The update is complete when the load balancer status is **ACTIVE**.

Additional resources

- loadbalancer in the *Command line interface reference*

CHAPTER 15. TROUBLESHOOTING AND MAINTAINING THE LOAD-BALANCING SERVICE

Basic troubleshooting and maintenance for the Load-balancing service (octavia) starts with being familiar with the OpenStack client commands for showing status and migrating instances, and knowing how to access logs. If you need to troubleshoot more in depth, you can SSH into one or more Load-balancing service instances (amphorae).

- Section 15.1, “Verifying the load balancer”
- Section 15.2, “Load-balancing service instance administrative logs”
- Section 15.3, “Migrating a specific Load-balancing service instance”
- Section 15.4, “Using SSH to connect to load-balancing instances”
- Section 15.5, “Showing listener statistics”
- Section 15.6, “Interpreting listener request errors”

15.1. VERIFYING THE LOAD BALANCER

You can troubleshoot the Load-balancing service (octavia) and its various components by viewing the output of the load balancer show and list commands.

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Verify the load balancer (**lb1**) settings.

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer show lb1
   ```

   Sample output

   ```
   +--------------------+-------------------------------------+
   | Field              | Value                               |
   +--------------------+-------------------------------------+
   | admin_state_up     | True                                |
   | created_at         | 2022-02-17T15:59:18                 |
   ```

```
| description         |                                      |
| flavor_id           | None                                 |
| id                  | 265d0b71-c073-40f4-9718-8a182c6d53ca |
| listeners           | 5aaa67da-350d-4125-9022-238e0f7b7f6f |
| name                | lb1                                  |
| operating_status    | ONLINE                               |
| pools               | 48f6664c-b192-4763-846a-da568354da4a |
| project_id          | 52376c9c5c2e434283266ae7cacd3a9c     |
| provider            | amphora                              |
| provisioning_status | ACTIVE                               |
| updated_at          | 2022-02-17T16:01:21                  |
| vip_address         | 192.0.2.177                          |
| vip_network_id      | afeaf55e-7128-4dff-80e2-98f8d1f2f44c |
| vip_port_id         | 94a12275-1505-4cdc-80c9-4432767a980f |
| vip_qos_policy_id   | None                                 |
| vip_subnet_id       | 06ffa90e-2b86-4fe3-9731-c7839b0be6de |
+---------------------+--------------------------------------+
```

3. Using the loadbalancer ID (**265d0b71-c073-40f4-9718-8a182c6d53ca**) from the previous step, obtain the ID of the amphora associated with the load balancer (**lb1**).

   Example

   ```
   $ openstack loadbalancer amphora list | grep 265d0b71-c073-40f4-9718-8a182c6d53ca
   ```

   Sample output

   ```
   | 1afabefd-ba09-49e1-8c39-41770aa25070 | 265d0b71-c073-40f4-9718-8a182c6d53ca |
   ALLOCATED | STANDALONE | 198.51.100.7  | 192.0.2.177   |
   ```

4. Using the amphora ID (**1afabefd-ba09-49e1-8c39-41770aa25070**) from the previous step, view amphora information.

   Example

   ```
   $ openstack loadbalancer amphora show 1afabefd-ba09-49e1-8c39-41770aa25070
   ```

   Sample output

   ```
   +-----------------+--------------------------------------+
   | Field           | Value                                |
   +-----------------+--------------------------------------+
   | id              | 1afabefd-ba09-49e1-8c39-41770aa25070 |
   | loadbalancer_id | 265d0b71-c073-40f4-9718-8a182c6d53ca |
   | compute_id      | ba9fc1c4-8aee-47ad-b47f-98f12ea7b200 |
   | lb_network_ip   | 198.51.100.7                         |
   | vrrp_ip         | 192.0.2.36                           |
   | ha_ip           | 192.0.2.177                          |
   | vrrp_port_id    | 07dcd894-487a-48dc-b0ec-7324fe5d2082 |
   | ha_port_id      | 94a12275-1505-4cdc-80c9-4432767a980f |
   | cert_expiration | 2022-03-19T15:59:23                  |
   | cert_busy       | False                                |
   | role            | STANDALONE                           |
   | status          | ALLOCATED                            |
   ```

```
| vrrp_interface  | None                                 |
| vrrp_id         | 1                                    |
| vrrp_priority   | None                                 |
| cached_zone     | nova                                 |
| created_at      | 2022-02-17T15:59:22                  |
| updated_at      | 2022-02-17T16:00:50                  |
| image_id        | 53001253-5005-4891-bb61-8784ae85e962 |
| compute_flavor  | 65                                   |
+-----------------+--------------------------------------+
```

5. View the listener (**listener1**) details.

Example

```
$ openstack loadbalancer listener show listener1
```

Sample output

```
+-----------------------------+--------------------------------------+
| Field                       | Value                                |
+-----------------------------+--------------------------------------+
| admin_state_up              | True                                 |
| connection_limit            | -1                                   |
| created_at                  | 2022-02-17T16:00:59                  |
| default_pool_id             | 48f6664c-b192-4763-846a-da568354da4a |
| default_tls_container_ref   | None                                 |
| description                 |                                      |
| id                          | 5aaa67da-350d-4125-9022-238e0f7b7f6f |
| insert_headers              | None                                 |
| l7policies                  |                                      |
| loadbalancers               | 265d0b71-c073-40f4-9718-8a182c6d53ca |
| name                        | listener1                            |
| operating_status            | ONLINE                               |
| project_id                  | 52376c9c5c2e434283266ae7cacd3a9c     |
| protocol                    | HTTP                                 |
| protocol_port               | 80                                   |
| provisioning_status         | ACTIVE                               |
| sni_container_refs          | []                                   |
| timeout_client_data         | 50000                                |
| timeout_member_connect      | 5000                                 |
| timeout_member_data         | 50000                                |
| timeout_tcp_inspect         | 0                                    |
| updated_at                  | 2022-02-17T16:01:21                  |
| client_ca_tls_container_ref | None                                 |
| client_authentication       | NONE                                 |
| client_crl_container_ref    | None                                 |
| allowed_cidrs               | None                                 |
+-----------------------------+--------------------------------------+
```

6. View the pool (**pool1**) and load-balancer members.

Example

```
$ openstack loadbalancer pool show pool1
```

Sample output

```
+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| admin_state_up       | True                                 |
| created_at           | 2022-02-17T16:01:08                  |
| description          |                                      |
| healthmonitor_id     | 4b24180f-74c7-47d2-b0a2-4783ada9a4f0 |
| id                   | 48f6664c-b192-4763-846a-da568354da4a |
| lb_algorithm         | ROUND_ROBIN                          |
| listeners            | 5aaa67da-350d-4125-9022-238e0f7b7f6f |
| loadbalancers        | 265d0b71-c073-40f4-9718-8a182c6d53ca |
| members              | b92694bd-3407-461a-92f2-90fb2c4aedd1 |
|                      | 4ccdd1cf-736d-4b31-b67c-81d5f49e528d |
| name                 | pool1                                |
| operating_status     | ONLINE                               |
| project_id           | 52376c9c5c2e434283266ae7cacd3a9c     |
| protocol             | HTTP                                 |
| provisioning_status  | ACTIVE                               |
| session_persistence  | None                                 |
| updated_at           | 2022-02-17T16:01:21                  |
| tls_container_ref    | None                                 |
| ca_tls_container_ref | None                                 |
| crl_container_ref    | None                                 |
| tls_enabled          | False                                |
+----------------------+--------------------------------------+
```

7. Verify HTTPS traffic flows across a load balancer whose listener is configured for **HTTPS** or **TERMINATED_HTTPS** protocols by connecting to the VIP address (**192.0.2.177**) of the load balancer.

TIP

Obtain the load-balancer VIP address by using the command, **openstack loadbalancer show <load_balancer_name>**.

NOTE

Security groups implemented for the load balancer VIP only allow data traffic for the required protocols and ports. For this reason you cannot ping load balancer VIPs, because ICMP traffic is blocked.

Example

```
$ curl -v https://192.0.2.177 --insecure
```

Sample output

```
* About to connect() to 192.0.2.177 port 443 (#0)
*   Trying 192.0.2.177...
* Connected to 192.0.2.177 (192.0.2.177) port 443 (#0)
* Initializing NSS with certpath: sql:/etc/pki/nssdb
* skipping SSL peer certificate verification
* SSL connection using TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
* Server certificate:
*  subject: CN=www.example.com,O=Dis,L=Springfield,ST=Denial,C=US
*  start date: Jan 15 09:21:45 2021 GMT
*  expire date: Jan 15 09:21:45 2021 GMT
*  common name: www.example.com
*  issuer: CN=www.example.com,O=Dis,L=Springfield,ST=Denial,C=US
> GET / HTTP/1.1
> User-Agent: curl/7.29.0
> Host: 192.0.2.177
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 30
<
* Connection #0 to host 192.0.2.177 left intact
```

Additional resources

- loadbalancer in the *Command line interface reference*

15.2. LOAD-BALANCING SERVICE INSTANCE ADMINISTRATIVE LOGS

The administrative log offloading feature of the Load-balancing service instance (amphora) covers all of the system logging inside the amphora except for the tenant flow logs. You can send tenant flow logs to the same syslog receiver where the administrative logs are sent. You can send tenant flow logs to the same syslog receiver that processes the administrative logs, but you must configure the tenant flow logs separately.

The amphora sends all administrative log messages by using the native log format for the application sending the message. The amphorae log to the Red Hat OpenStack Platform (RHOSP) Controller node in the same location as the other RHOSP logs (**/var/log/containers/octavia-amphorae/**).

Additional resources

- Chapter 5, *Managing Load-balancing service instance logs*

15.3. MIGRATING A SPECIFIC LOAD-BALANCING SERVICE INSTANCE

In some cases you must migrate a Load-balancing service instance (amphora). For example, if the host is being shut down for maintenance

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. Locate the ID of the amphora that you want to migrate. You need to provide the ID in a later step.

```
$ openstack loadbalancer amphora list
```

3. To prevent the Compute scheduler service from scheduling any new amphorae to the Compute node being evacuated, disable the Compute node (**compute-host-1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   ### Example

   ```
   $ openstack compute service set compute-host-1 nova-compute --disable
   ```

4. Fail over the amphora by using the amphora ID (**ea17210a-1076-48ff-8a1f-ced49ccb5e53**) that you obtained.

   ### Example

   ```
   $ openstack loadbalancer amphora failover ea17210a-1076-48ff-8a1f-ced49ccb5e53
   ```

Additional resources

- compute service set in the *Command line interface reference*
- loadbalancer in the *Command line interface reference*

15.4. USING SSH TO CONNECT TO LOAD-BALANCING INSTANCES

Use SSH to log in to Load-balancing service instances (amphorae) when troubleshooting service problems.

It can be helpful to use Secure Shell (SSH) to log into running Load-balancing service instances (amphorae) when troubleshooting service problems.

Prerequisites

- You must have the Load-balancing service (octavia) SSH private key.

Procedure

1. On the director node, start **ssh-agent** and add your user identity key to the agent:

   ```
   $ eval $(ssh-agent -s)
   $ sudo -E ssh-add /etc/octavia/ssh/octavia_id_rsa
   ```

2. Source your credentials file.

   ### Example

   ```
   $ source ~/overcloudrc
   ```

3. Determine the IP address on the load-balancing management network (**lb_network_ip**) for the amphora that you want to connect to:

   ```
   $ openstack loadbalancer amphora list
   ```

4. Use SSH to connect to the amphora:

   ```
   $ ssh -A -t tripleo-admin@<controller_node_IP_address> ssh cloud-user@<lb_network_ip>
   ```

5. When you are finished, close your connection to the amphora and stop the SSH agent:

   ```
   $ exit
   ```

Additional resources

- loadbalancer in the *Command line interface reference*

15.5. SHOWING LISTENER STATISTICS

Using the OpenStack Client, you can obtain statistics about the listener for a particular Red Hat OpenStack Platform (RHOSP) loadbalancer:

- current active connections (**active_connections**).
- total bytes received (**bytes_in**).
- total bytes sent (**bytes_out**).
- total requests that were unable to be fulfilled (**request_errors**).
- total connections handled (**total_connections**).

Procedure

1. Source your credentials file.

   Example

   ```
   $ source ~/overcloudrc
   ```

2. View the stats for the listener (**listener1**).

   NOTE

   Values inside parentheses are sample values that are used in the example commands in this procedure. Substitute these sample values with values that are appropriate for your site.

   Example

   ```
   $ openstack loadbalancer listener stats show listener1
   ```

TIP

If you do not know the name of the listener, enter the command **loadbalancer listener list**.

Sample output

```
+--------------------+-------+
| Field              | Value |
+--------------------+-------+
| active_connections | 0     |
| bytes_in           | 0     |
| bytes_out          | 0     |
| request_errors     | 0     |
| total_connections  | 0     |
+--------------------+-------+
```

Additional resources

- loadbalancer listener stats show in the *Command line interface reference*
- Section 15.6, “Interpreting listener request errors”

15.6. INTERPRETING LISTENER REQUEST ERRORS

You can obtain statistics about the listener for a particular Red Hat OpenStack Platform (RHOSP) loadbalancer. For more information, see Section 15.5, “Showing listener statistics” .

One of the statistics tracked by the RHOSP loadbalancer, **request_errors**, is only counting errors that occurred in the request from the end user connecting to the load balancer. The **request_errors** variable is not measuring errors reported by the member server.

For example, if a tenant connects through the RHOSP Load-balancing service (octavia) to a web server that returns an HTTP status code of **400 (Bad Request)**, this error is not collected by the Load-balancing service. Loadbalancers do not inspect the content of data traffic. In this example, the loadbalancer interprets this flow as successful because it transported information between the user and the web server correctly.

The following conditions can cause the **request_errors** variable to increment:

- early termination from the client, before the request has been sent.
- read error from the client.
- client timeout.
- client closed the connection.
- various bad requests from the client.

Additional resources

- loadbalancer listener stats show in the *Command line interface reference*
- Section 15.5, “Showing listener statistics”