# Red Hat OpenStack Platform 13

## Using Octavia for Load Balancing-as-a-Service

Information about how to load balance network traffic on the data plane

Last Updated: 2021-09-23

# Red Hat OpenStack Platform 13 Using Octavia for Load Balancing-as-a-Service

Information about how to load balance network traffic on the data plane

OpenStack Team
rhos-docs@redhat.com

## Legal Notice



## Abstract

A guide for installing, configuring, operating, troubleshooting, and upgrading the 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.

## Using the Direct Documentation Feedback (DDF) function

Use the **Add Feedback** DDF function for direct comments on specific sentences, paragraphs, or code blocks.

1. View the documentation in the *Multi-page HTML* format.
2. Ensure that you see the **Feedback** button in the upper right corner of the document.
3. Highlight the part of text that you want to comment on.
4. Click **Add Feedback**.
5. Complete the **Add Feedback** field with your comments.
6. Optional: Add your email address so that the documentation team can contact you for clarification on your issue.
7. Click **Submit**.

# CHAPTER 1. UNDERSTANDING 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.

This section introduces the Load-balancing service, explains how load balancing integrates with the rest of RHOSP, and describes the various load-balancing components.

## 1.1. WHAT IS 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 a fleet of virtual machines, containers, or bare metal servers—collectively known as *amphorae*--which it spins up on demand. The ability to provide on-demand, horizontal scaling makes the Load-balancing service a fully-featured load balancer and well-suited for large RHOSP, enterprise deployments.

**NOTE**

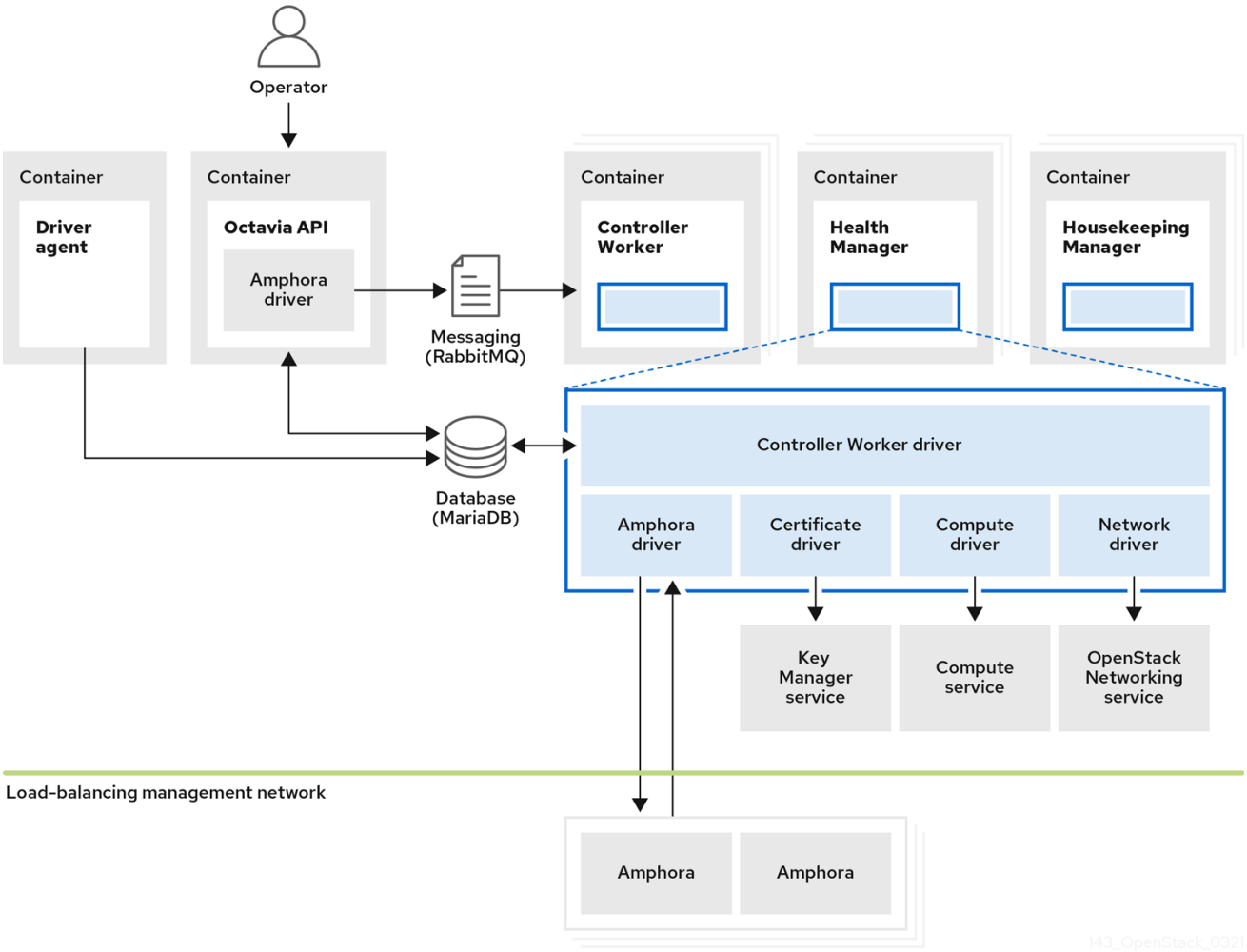
Red Hat does not support a migration path from Neutron-LBaaS to octavia. However, there are some unsupported open source tools that are available. See "Additional resources," later.

**Additional resources**

- nlbaas2octavia-lb-replicator on GitHub
- Load-balancing service components
- Load-balancing service object model
- Where does Load-balancing fit in OpenStack?

## 1.2. 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**).




The various 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:

**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 should an amphora encounter a failure.

**Housekeeping Manager (octavia_housekeeping container)**

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

**Driver agent (octavia_driver_agent container)**

Supports other provider drivers, such as OVN.

**Amphora**

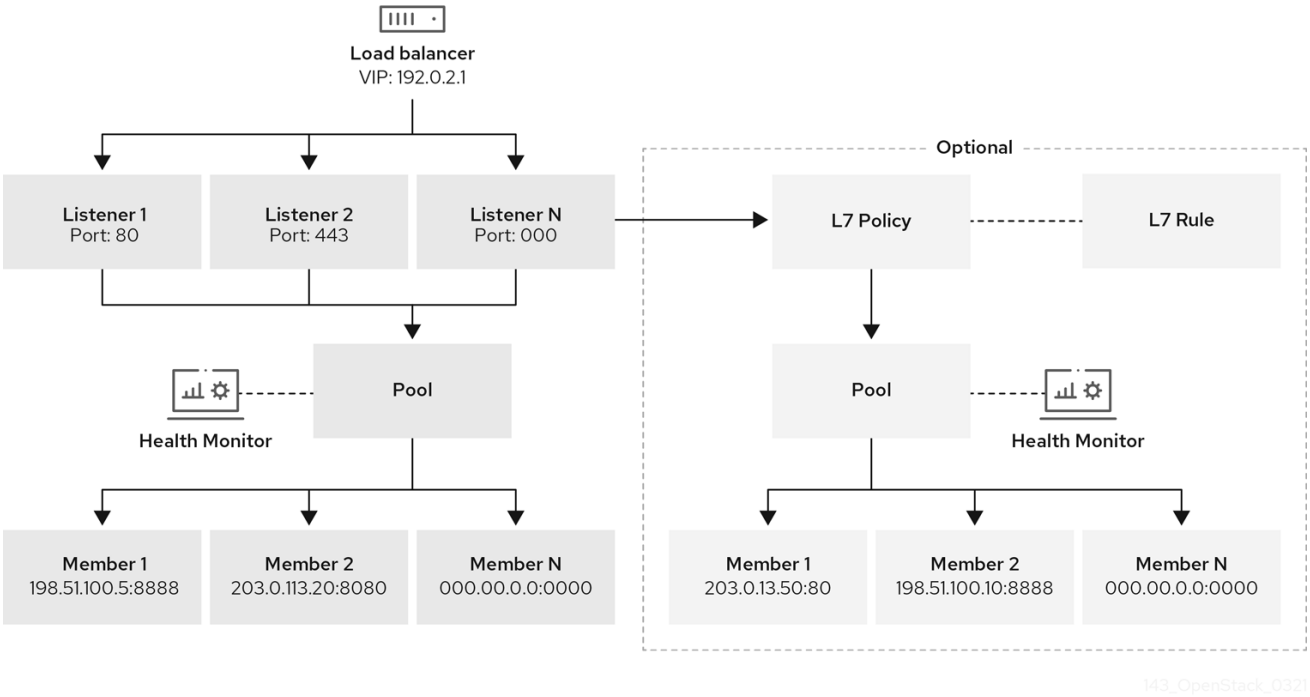
The VM instance that performs the load balancing. Amphorae are typically instances running on Compute nodes, and are configured with load balancing parameters according to the listener, pool, health monitor, L7 policies, and members' configuration. Amphorae send a periodic heartbeat to the Health Manager.

### Additional resources

- What is the Load-balancing service?
- Load-balancing service object model
- Where does Load-balancing fit in OpenStack?

## 1.3. LOAD-BALANCING SERVICE OBJECT MODEL

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




**Load balancer**

The top API object that represents the load-balancing entity. The VIP address is allocated when the load balancer is created. When creating the load balancer using the amphora provider, 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 does periodic health checks on each back end member server to pre-emptively detect failed servers and temporarily pull them out of the pool.

**Pool**

A group of members that handle client requests from the load balancer (amphora). Pools can be associated with more than one listener using the API. Pools can also be shared 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 used to determine whether an L7 policy should be applied 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 should be taken by the load balancer if all of the rules in the policy are true.

**Additional resources**

- What is the Load-balancing service?
- Load-balancing service components
- Where does Load-balancing fit in OpenStack?

## 1.4. WHERE DOES LOAD-BALANCING FIT 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 for various functions:

- **Compute service (nova)** - For managing the Load-balancing service VM instance (amphora) lifecycle, and spinning up compute resources on demand.
- **Networking service (neutron)** - For network connectivity between Load-balancing service instances (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 octavia to authenticate with other RHOSP services.
- **Image service (glance)** - For storing the amphora virtual machine image.
- **Common Libraries (oslo)** - For communication between octavia controller components, making octavia work within the standard OpenStack framework and review system, and project code structure.
- **Taskflow** - Is technically part of oslo; however, octavia makes extensive use of this job flow system when orchestrating back-end service configuration and management.

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

**Additional resources**

- What is the Load-balancing service?
- Load-balancing service components
- Load-balancing service object model

# CHAPTER 2. PLANNING FOR THE LOAD-BALANCING SERVICE

You must perform several tasks when planning for deploying the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia):

- Choosing which provider to use with the Load-balancing service, or choosing to enable multiple providers simultaneously.
- Understanding what software the Load-balancing service requires.
- Ensuring that your RHOSP undercloud meets the necessary prerequisites.
- Deciding to make the Load-balancing service highly available.

## 2.1. LOAD-BALANCING SERVICE PROVIDER DRIVERS

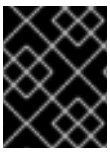
The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) supports enabling multiple provider drivers using the Octavia v2 API. You can choose to use one provider driver, or multiple provider drivers simultaneously.

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

Amphora, the default, is a highly available load balancer with a rich 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. Typically used for east-west, layer 4 network traffic, OVN provisions fast and consumes less 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 (octavia), without requiring additional installation or configuration steps. As with all RHOSP deployments, the default load-balancing provider driver, amphora, remains enabled and fully supported.



**IMPORTANT**

The procedures in this section *apply only to the amphora load-balancing provider* , unless indicated otherwise.

**Additional resources**

- Load-balancing service feature support matrix
- Load-balancing service software requirements
- Load-balancing service prerequisites for the undercloud
- Basics of active-standby topology for Load-balancing service instances
- Post-deployment steps for the Load-balancing service

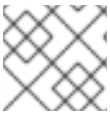
## 2.2. LOAD-BALANCING SERVICE (OCTAVIA) FEATURE SUPPORT MATRIX

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 with a rich feature set but 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 less load-balancing features than amphora.

The following table lists features in octavia that Red Hat OpenStack Platform (RHOSP) 13 supports and in which maintenance release support for the feature began.



**NOTE**

If the feature is not listed, then RHOSP 13 does not support the feature.

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

| Feature | Support level in RHOSP 13 releases | |
|---|---|---|
| | **Amphora Provider** | **OVN Provider** |
| ML2/OVS L3 HA | Full support—13.0 and all maintenance releases | Not applicable |
| ML2/OVS DVR | Full support—29 August 2018 maintenance release and later | Not applicable |
| ML2/OVS L3 HA + composable network node [1] | Full support—13 March 2019 maintenance release and later | Not applicable |
| ML2/OVS DVR + composable network node [1] | Full support—13 March 2019 maintenance release and later | Not applicable |
| ML2/OVN L3 HA | Full support—29 August 2018 maintenance release and later | Full support—28 October 2020 maintenance release and later |
| ML2/OVN DVR | Full support—13 November 2018 maintenance release and later | Full support—28 October 2020 maintenance release and later |
| ML2/ODL | Full support—16 January 2019 maintenance release and later | Not applicable |
| Amphora active-standby | Full support—28 October 2020 maintenance release and later | Not supported |
| Terminated HTTPS load balancers | Full support—10 March 2020 maintenance release and later | Not supported |

| | | |
|---|---|---|
| Amphora spare pool | Technology Preview only—30 April 2019 maintenance release and later | Not applicable |
| UDP | Full support—28 October 2020 maintenance release and later | Full support—28 October 2020 maintenance release and later |

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

**Additional resources**

- Load-balancing service provider drivers
- Load-balancing service software requirements
- Load-balancing service prerequisites for the undercloud
- Basics of active-standby topology for Load-balancing service instances
- Post-deployment steps for the Load-balancing service

## 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

**Additional resources**

- Load-balancing service provider drivers
- Load-balancing service feature support matrix
- Load-balancing service prerequisites for the undercloud
- Basics of active-standby topology for Load-balancing service instances
- Post-deployment steps for the Load-balancing service

## 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:

- your undercloud is already installed and ready to deploy an overcloud with the Load-balancing service (octavia) enabled.
- only container deployments are supported.
- the Load-balancing service runs on your Controller node.

**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. To prepare the undercloud, see the *Transitioning to Containerized Services* guide.

**Additional resources**

- Load-balancing service provider drivers
- Load-balancing service feature support matrix
- Load-balancing service software requirements
- Basics of active-standby topology for Load-balancing service instances
- Post-deployment steps for the Load-balancing service

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

When deploying 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 are able to 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. Should the active instance become unhealthy, the instance automatically fails over to the standby instance making it active. The Load-balancing service health manager automatically rebuilds an instance that has failed.

**Additional resources**

- Enabling Amphora active-standby topology
- Load-balancing service provider drivers

- Load-balancing service feature support matrix
- Load-balancing service software requirements
- Load-balancing service prerequisites for the undercloud
- Post-deployment steps for the Load-balancing service

## 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 Octavia Controller worker and health manager.

**NOTE**

Do not modify RHOSP heat templates directly. Create a custom environment file (for example, **octavia-environment.yaml**) to override default parameter values.

### Amphora Image

On pre-provisioned servers, you must install the amphora image on the undercloud before deploying 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

- Environment Files in the *Advanced Overcloud Customization* guide
- Including Environment Files in Overcloud Creation in the *Advanced Overcloud Customization* guide
- Load-balancing service provider drivers
- Load-balancing service feature support matrix
- Load-balancing service software requirements

- Load-balancing service prerequisites for the undercloud
- Basics of active-standby topology for Load-balancing service instances

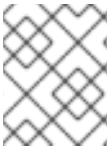
# CHAPTER 3. SECURING THE LOAD-BALANCING SERVICE

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

This section gives an overview of the TLS handshake, briefly discusses the certificate lifecycle and intermediate certificate chains, and provides steps for sites that want to use their own certificates instead of the automatically generated ones provided by Red Hat OpenStack Platform.

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

The Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) controller processes communicate with Load-balancing service instances (amphorae) over a TLS connection much like an HTTPS connection to a website. However, the Load-balancing service validates that both sides are trusted by doing a two-way TLS authentication.



**NOTE**

This is a simplification of the full TLS handshake process. See TLS 1.3 RFC 8446 for the full handshake.

There are two phases involved in two-way TLS authentication. *Phase one* is when 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 it 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 the second phase.

*Phase two* is when 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**

- Certificate lifecycles for the Load-balancing service
- Configuring Load-balancing service certificates and keys

## 3.2. CERTIFICATE LIFECYCLES FOR THE LOAD-BALANCING SERVICE

Using the server certificate authority certificates and keys, the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) controller uniquely generates 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**

- Two-way TLS authentication in the Load-balancing service

- Configuring Load-balancing service certificates and keys

## 3.3. CONFIGURING LOAD-BALANCING SERVICE CERTIFICATES AND KEYS

You can allow Red Hat OpenStack Platform (RHOSP) director to generate certificates and keys, or you can supply your own. We recommend that you allow 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**

One benefit of using RHOSP director to generate certificates and keys for you, is that director automatically renews certificates before they expire. If you decide to use your own certificates, remember that director cannot automatically renew them.

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," later.)

**Procedure**

1. On the machine on which you run the **openstack overcloud deploy** command, create a YAML custom environment file.

   **Example**

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

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

   - **OctaviaCaCert**:
     The certificate for the CA Octavia will use 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]
```

TIP

YAML files are extremely sensitive about where in the file a parameter is placed. Make sure that **parameter_defaults:** starts in the first column (no leading whitespace characters), and your parameter value pairs start in column five (each parameter has four whitespace characters in front of it).

3. 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**

- Changing Load-balancing service default settings
- Environment Files in the *Advanced Overcloud Customization* guide
- Including Environment Files in Overcloud Creation in the *Advanced Overcloud Customization* guide
- Two-way TLS authentication in the Load-balancing service
- Certificate lifecycles for the Load-balancing service

# CHAPTER 4. DEPLOYING THE LOAD-BALANCING SERVICE

This section discusses how to deploy the Red Hat OpenStack Platform Load-balancing service (octavia) and how to change its configuration. Also included is a discussion about how to make Load-balancing service instances (amphorae) highly available.

## 4.1. DEPLOYING THE LOAD-BALANCING SERVICE

You deploy the Load-balancing service (octavia) using Red Hat OpenStack Platform (RHOSP) *director*. A toolset for installing and managing a complete OpenStack environment, 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 the Load-balancing service and all of your OpenStack environment.

**Prerequisites**

- Ensure that your environment has access to the octavia image.
- Read and understand the Director Installation and Usage Guide.

**Procedure**

- Run the RHOSP director command, **openstack overcloud deploy** 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 amphora image to the latest amphora image during a stack update or upgrade.

**Additional resources**

- Post-deployment steps for the Load-balancing service
- Creating the Overcloud with the CLI Tools in the *Director Installation and Usage* guide
- Changing Load-balancing service default settings

## 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 through a particular Red Hat OpenStack Platform (RHOSP) Orchestration service (heat) parameter, **OctaviaLoadBalancerTopology**. You set the value of **OctaviaLoadBalancerTopology** in a *custom environment file*, which is a special type of template that provides customization for your heat templates.

IMPORTANT

As a best practice, always make Load-balancing service configuration changes to a custom environment file and run the **openstack overcloud deploy** command. You risk losing your configuration changes when you do not use director and manually change values in Load-balancing service configuration files.

**Prerequisites**

- Read and understand the Advanced Overcloud Customization guide.
- Anti-affinity must be enabled for the Compute service. (This is the default.)
- A minimum of three Compute node hosts are required: two Compute node hosts to place the amphorae on different hosts (Compute anti-affinity), and a third host to successfully fail over an active-standby load balancer.

**Procedure**

1. On the undercloud host, logged in as the stack user, create a custom YAML environment file.

   **Example**

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

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

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

   TIP

   YAML files are extremely sensitive about where in the file a parameter is placed. Make sure that **parameter_defaults:** starts in the first column (no leading whitespace characters), and your parameter value pairs start in column five (each parameter has four whitespace characters in front of it). Consider using a YAML validator to ensure that your YAML syntax is valid.

3. 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
   ```

**Verification steps**

**Verification steps**

- 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 *Advanced Overcloud Customization* guide
- Including Environment Files in Overcloud Creation in the *Advanced Overcloud Customization* guide
- Basics of active-standby topology for Load-balancing service instances

## 4.3. CHANGING LOAD-BALANCING SERVICE DEFAULT SETTINGS

You make configuration changes to the Load-balancing service (octavia) with Red Hat OpenStack Platform (RHOSP) *director*. A toolset for installing and managing a complete OpenStack environment, director uses Orchestration service (heat) templates that are a set of plans for your environment. You can customize aspects of the overcloud with a *custom environment file*, which is a special type of template that provides customization for your heat templates.

**IMPORTANT**

As a best practice, always make Load-balancing service configuration changes to a custom environment file and run the **openstack overcloud deploy** command. You risk losing your configuration changes when you do not use director and manually change values in Load-balancing service configuration files.

**Prerequisites**

- Read and understand the Advanced Overcloud Customization guide.
- 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 for the neutron network used for the load balancer management network.
  - **OctaviaControlSubnetCidr**
    The subnet for amphora control subnet in CIDR form.
  - **OctaviaMgmtPortDevName**

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

**Procedure**

1. On the machine on which you run the **openstack overcloud deploy** command, create a YAML custom environment file.

   **Example**

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

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

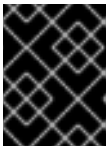
   **Example**

   ```
   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'
   ```

   **TIP**

   YAML files are extremely sensitive about where in the file a parameter is placed. Make sure that **parameter_defaults:** starts in the first column (no leading whitespace characters), and your parameter value pairs start in column five (each parameter has four whitespace characters in front of it).

3. 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**

- Environment files in the *Advanced Overcloud Customization* guide

- Including Environment Files in Overcloud Creation in the *Advanced Overcloud Customization* guide
- Deploying the Load-balancing service

# CHAPTER 5. CONFIGURING LOAD-BALANCING SERVICE FLAVORS

Load-balancing service (octavia) *flavors* are predefined sets of provider configuration options that you define, so that users can easily create the type of load balancer that fulfills their need.

This section explains more about flavors and provides the steps that you must follow to create them.

## 5.1. ABOUT LOAD-BALANCING SERVICE FLAVORS

Load-balancing service (octavia) *flavors* are predefined 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.

There are three steps required 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.

**Additional resources**

- Listing Load-balancing service provider capabilities
- Defining flavor profiles
- Creating Load-balancing service flavors

## 5.2. LISTING LOAD-BALANCING SERVICE PROVIDER CAPABILITIES

The first step in defining a Load-balancing service (octavia) flavor, is to review the list of capabilities that each provider driver exposes. You use obtain the list of provider driver capabilities by using the OpenStack client.

**Prerequisites**

- You must have OpenStack administrator privileges.

**Procedure**

1. Log in as the stack user and run the following OpenStack client command:

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

**Example**

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

The command output lists all of the capabilities 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.                          |
| ...                  | ...                                                 |
+----------------------+-----------------------------------------------------+
```

2. Note the names of the capabilities that you want in the flavor that you are creating.

**Additional resources**

- About Load-balancing service flavors
- Defining flavor profiles
- Creating Load-balancing service flavors

## 5.3. DEFINING FLAVOR PROFILES

The second step in defining a Load-balancing service (octavia) flavor, is to define a profile for the flavor. Profiles include the provider name and a list of capabilities. You create a flavor profile by using the OpenStack client.

**Prerequisites**

- You must have OpenStack administrator privileges.
- You must know which load-balancing provider and which of its capabilities will go into the flavor profile.

**Procedure**

- Log in as the stack user and run the following OpenStack client command:

```
$ 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 for the amphora provider has been defined. When this flavor is used, a single amphora load balancer is created.

**Verification steps**

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

**Additional resources**

- About Load-balancing service flavors
- Listing Load-balancing service provider capabilities
- Creating Load-balancing service flavors

## 5.4. CREATING LOAD-BALANCING SERVICE FLAVORS

The final step in defining a Load-balancing service (octavia) flavor is to create the actual user-facing flavor. The name of the flavor is the value users specify when they create a load balancer. Like the other steps, you use the OpenStack client to create a flavor.

**Prerequisites**

- You must have OpenStack administrator privileges.
- You must have created a flavor profile.

**Procedure**

- Log in as the stack user and run the following OpenStack client command:

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

**TIP**

We recommend that you provide a detailed description for users to clearly understand the capabilities of the flavor that you are providing.

**Example**

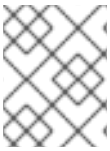
```
$ openstack loadbalancer flavor create --name standalone-lb --flavorprofile amphora-single-profile --description "A non-high availability load balancer for testing."
```

**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 will 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 they will not be able to create a load balancer using the disabled flavor.

**Additional resources**

- About Load-balancing service flavors
- Listing Load-balancing service provider capabilities
- Defining flavor profiles

# CHAPTER 6. 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.

## 6.1. 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-balancer management network*. Hosts that run the Load-balancing service must have interfaces to connect to the load balancer 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 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.

**Additional resources**

- Load-balancing service instance monitoring
- Load-balancing service pool member monitoring
- Load balancer monitoring
- Load balancer functionality monitoring

## 6.2. 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. Anytime 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.

**Additional resources**

- Load-balancing mangement network
- Load-balancing service pool member monitoring
- Load balancer monitoring
- Load balancer functionality monitoring

## 6.3. 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 should poll for health information in regular intervals.

**Additional resources**

- Load-balancing mangement network
- Load-balancing service instance monitoring
- Load balancer monitoring
- Load balancer functionality monitoring

## 6.4. LOAD BALANCER MONITORING

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

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

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

**Additional resources**

- Load-balancing mangement network
- Load-balancing service instance monitoring

- Load-balancing service pool member monitoring
- Load balancer functionality monitoring

## 6.5. LOAD BALANCER FUNCTIONALITY MONITORING

You can monitor the operational status of your load balancer using the **openstack loadbalancer status show** command. It reports the current operation status of the load balancer and its child objects.

You might also want to use an external monitoring service that connects to your load balancer listeners and monitors them from outside of the cloud. This type of monitoring 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.

**Additional resources**

- Load-balancing mangement network
- Load-balancing service instance monitoring
- Load-balancing service pool member monitoring
- Load balancer monitoring

## 6.6. 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 are performing 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.
  Your back end servers must be configured to allow PINGs in order for these health checks to pass.

**IMPORTANT**

A PING health monitor only checks 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. Only use PING health monitors 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 should open a TCP connection and, after the TCP handshake, close the connection without sending any data.
- **UDP-CONNECT**: do 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.

**Additional resources**

- Creating Load-balancing service health monitors
- Modifying Load-balancing service health monitors
- Deleting Load-balancing service health monitors
- Load-balancing service health monitor configuration arguments
- Best practices for Load-balancing service HTTP health monitors

## 6.7. CREATING LOAD-BALANCING SERVICE HEALTH MONITORS

Load-balancing service (octavia) health monitors can help you avoid service disruptions for your users, because the monitors run periodic health checks on each back end server to pre-emptively detect failed servers and temporarily pull them out of the pool. You can use the OpenStack client to create a health monitor.

**Procedure**

- Log in and run the following OpenStack client command, **openstack loadbalancer healthmonitor create**.

  **Example**

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

**Verification steps**

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

**Additional resources**

- loadbalancer healthmonitor create command in the *Command Line Interface Reference*

- About Load-balancing service health monitors
- Modifying Load-balancing service health monitors
- Deleting Load-balancing service health monitors
- Load-balancing service health monitor configuration arguments
- Best practices for Load-balancing service HTTP health monitors

## 6.8. MODIFYING LOAD-BALANCING SERVICE HEALTH MONITORS

You can use the OpenStack client command **openstack loadbalancer healthmonitor set** to modify configuration settings for Load-balancing service (octavia) health monitors.

### Procedure

- Log in and run the following OpenStack client command, **openstack loadbalancer healthmonitor set**.

  #### Example

  ```
  $ openstack loadbalancer healthmonitor set my-health-monitor --delay 600
  ```

### Verification steps

- Running the OpenStack client command, **openstack loadbalancer healthmonitor show** displays certain details for a health monitor.

### Additional resources

- loadbalancer healthmonitor set command in the *Command Line Interface Reference*
- About Load-balancing service health monitors
- Creating Load-balancing service health monitors
- Deleting Load-balancing service health monitors
- Load-balancing service health monitor configuration arguments
- Best practices for Load-balancing service HTTP health monitors

## 6.9. DELETING LOAD-BALANCING SERVICE HEALTH MONITORS

If you ever need to remove a Load-balancing service (octavia) health monitor, you can use the OpenStack client.

### TIP

An alternative to deleting a health monitor is to disable it. See **loadbalancer healthmonitor set** in "Additional resources," later.

### Procedure

- Log in and run the following OpenStack client command, **openstack loadbalancer healthmonitor delete**.

  Example

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

Verification steps

- Run the OpenStack client command, **openstack loadbalancer healthmonitor list** and verify that your health monitor no longer exists.

Additional resources

- loadbalancer healthmonitor delete command in the *Command Line Interface Reference*
- loadbalancer healthmonitor set command in the *Command Line Interface Reference*
- About Load-balancing service health monitors
- Creating Load-balancing service health monitors
- Modifying Load-balancing service health monitors
- Load-balancing service health monitor configuration arguments
- Best practices for Load-balancing service HTTP health monitors

## 6.10. LOAD-BALANCING SERVICE HEALTH MONITOR CONFIGURATION ARGUMENTS

*All* health monitor types for the Load-balancing service (octavia) require the following configurable arguments:

- **--delay**: Number of seconds to wait between health checks.
- **--timeout**: Number of seconds to wait for any given health check to complete. **timeout** should 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 to the earlier listed arguments, 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 should be 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 just **200**.

**Additional resources**

- loadbalancer healthmonitor create command in the *Command Line Interface Reference*
- About Load-balancing service health monitors
- Creating Load-balancing service health monitors
- Modifying Load-balancing service health monitors
- Deleting Load-balancing service health monitors
- Best practices for Load-balancing service HTTP health monitors

## 6.11. BEST PRACTICES FOR LOAD-BALANCING SERVICE HTTP HEALTH MONITORS

Please keep the following best practices in mind when writing the code that generates the health check in your web application:

- The health monitor **url-path** should not require authentication to load.
- By default, the health monitor **url-path** should return an **HTTP 200 OK** status code to indicate a healthy server unless you specify alternate **expected-codes**.
- The health check should do 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 should return in a sub-second interval.
  - It should not induce significant load on the application server.
- The page generated by the health check should never be cached, though the code running 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 generating the page at the health monitor **url-path** would incorporate 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.

**Additional resources**

- About Load-balancing service health monitors
- Creating Load-balancing service health monitors

- Modifying Load-balancing service health monitors
- Deleting Load-balancing service health monitors
- Load-balancing service health monitor configuration arguments

# CHAPTER 7. CREATING NON-SECURE HTTP LOAD BALANCERS

This section describes how to create various types of load balancers for non-secure HTTP network traffic.

## 7.1. CREATING AN HTTP LOAD BALANCER WITH A HEALTH MONITOR

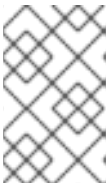
You can use the OpenStack client to create a load balancer to manage network traffic for non-secure HTTP applications. This is a basic solution for networks that are not compatible with OpenStack Networking floating IPs, such as IPv6 networks. It is a best practice to also create a health monitor to ensure that your back end members remain available.

### Prerequisites

- A private subnet that contains back end servers that host non-secure HTTP applications on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.

### Procedure

1. 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 ones that are appropriate for your site.

**Example**

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

2. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next step.

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
   ```

   When you see a status of **ACTIVE**, the listener is created and you can go to the next step.

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 on the pool (**pool1**) that connects to the back end servers and tests the path (/).

   **Example**

   ```
   $ openstack loadbalancer healthmonitor create --delay 5 --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**

   ```
   $ 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 steps**

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   **Example**

   ```
   $ openstack loadbalancer show lb1
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | 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          | 2021-01-15T11:12:13                  |
| 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 (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   **Example**

   ```
   $ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:16:23                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                |                                      |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2021-01-15T11:20:45                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

**Additional resources**

- loadbalancer in the *Command Line Interface Reference*
- Creating an HTTP load balancer using a floating IP
- Creating an HTTP load balancer with session persistence

## 7.2. CREATING AN HTTP LOAD BALANCER USING A FLOATING IP

To manage network traffic for non-secure HTTP applications, you can use the OpenStack client to 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 should 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

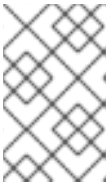
Floating IPs do not work with IPv6 networks.

**Prerequisites**

- A private subnet that contains back end servers that host non-secure HTTP applications on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A floating IP that can be used with a load balancer VIP.
- An OpenStack Networking shared external (public) subnet that is reachable from the internet that is used for the floating IP.

**Procedure**

1. 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 ones that are appropriate for your site.

**Example**

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

2. Note the value of **load_balancer_vip_port_id**, as you will need to provide it in a later step.
3. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next 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

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

6. Create a health monitor on the pool (**pool1**) that connects to the back end servers and tests the path (/).

   Example

   ```
   $ openstack loadbalancer healthmonitor create --delay 5 --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

   ```
   $ 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
   ```

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

   Example

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

9. Note the value of **floating_ip_address**, as you will need to provide it in the next 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 steps

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

   Example

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

   You should see output similar to the following:

   ```
   * 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 (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   Example

   ```
   $ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.02.10                          |
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:23                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                |                                      |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2021-01-15T11:28:42                  |
   | 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*
- Creating an HTTP load balancer with a health monitor
- Creating an HTTP load balancer with session persistence

## 7.3. CREATING AN HTTP LOAD BALANCER WITH SESSION PERSISTENCE

To manage network traffic for non-secure HTTP applications, you can use the OpenStack client to 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, and provides users with a good experience.

### Prerequisites

- A private subnet that contains back end servers that host non-secure HTTP applications on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.
- The non-secure web applications whose network traffic you are load balancing have cookies enabled.

### Procedure

1. 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 ones that are appropriate for your site.

**Example**

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

2. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next step.

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**

```
$ 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 on the pool (**pool1**) that connects to the back end servers and tests the path (/).

   Example

```
$ openstack loadbalancer healthmonitor create --delay 5 --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

```
$ 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 steps

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   Example

```
$ openstack loadbalancer show lb1
```

   You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:58                  |
| 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          | 2021-01-15T11:28:42                  |
| 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 (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   **Example**

   ```
   $ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.02.10                          |
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:23                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                |                                      |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2021-01-15T11:28:42                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

**Additional resources**

- loadbalancer in the *Command Line Interface Reference*
- Creating an HTTP load balancer with a health monitor
- Creating an HTTP load balancer using a floating IP

# CHAPTER 8. CREATING SECURE HTTP LOAD BALANCERS

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

## 8.1. 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. One disadvantage of non-terminated HTTPS load balancers is that they do not support advanced load balancer features like Layer 7 functionality.

### Additional resources

- Creating a non-terminated HTTPS load balancer

## 8.2. 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 private subnet that contains back end servers that host HTTPS applications that have been configured with a TLS-encrypted web application on TCP port 443.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.

### Procedure

1. 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 ones that are appropriate for your site.

**Example**

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

2. Monitor the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next step.

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**

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

5. Create a health monitor on the pool (**pool1**) that connects to the back end servers and tests the path (*/*).

   **Example**

   ```
   $ openstack loadbalancer healthmonitor create --delay 5 --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**

   ```
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.10 --
   protocol-port 443 pool1
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.11 --
   protocol-port 443 pool1
   ```

**Verification steps**

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   **Example**

   ```
   $ openstack loadbalancer show lb1
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | 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          | 2021-01-15T11:12:42                  |
| 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.
   A working member (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   **Example**

   ```
   $ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                |                                      |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 443                                  |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2021-01-15T11:12:42                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

**Additional resources**

- Manage Secrets with OpenStack Key Manager guide.
- loadbalancer in the *Command Line Interface Reference*
- About non-terminated HTTPS load balancers

## 8.3. 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. By terminating 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.

**Additional resources**

- Creating a TLS-terminated HTTPS load balancer
- Creating a TLS-terminated HTTPS load balancer with SNI
- Creating HTTP and TLS-terminated HTTPS load balancing on the same IP and back end

## 8.4. CREATING A TLS-TERMINATED HTTPS LOAD BALANCER

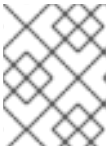
By using TLS-terminated HTTPS load balancers, you offload the CPU-intensive encryption operations to the load balancer, and allow the load balancer to use advanced features such as Layer 7 inspection. It is a best practice to also create a health monitor to ensure that your back end members remain available.

**Prerequisites**

- A private subnet that contains back end servers that host non-secure HTTP applications on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.
- TLS public-key cryptography has been configured with the following characteristics:
  - A TLS certificate, key, and 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 key is not encrypted with a passphrase.
  - 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). (See link to the *Manage Secrets with OpenStack Key Manager guide* in "Additional resources" later in this topic.)

**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

The values in parentheses are provided as examples. Replace these with values appropriate for your site.

Example

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

2. Using the Key Manager service, 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)"
   ```

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

   Example

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

4. Monitor the state of the load balancer.

   Example

   ```
   $ openstack loadbalancer show lb1
   ```

   When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next step.

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 --name listener1 --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

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

7. Create a health monitor on the pool (**pool1**) that connects to the back end servers and tests the path (/).

   Example

```
$ openstack loadbalancer healthmonitor create --delay 5 --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**

```
$ 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 steps**

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   **Example**

```
$ openstack loadbalancer show lb1
```

   You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| 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          | 2021-01-15T11:12:42                  |
| 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.
   A working member (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   **Example**

```
$ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
```

You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                |                                      |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 80                                   |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2021-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

**Additional resources**

- Manage Secrets with OpenStack Key Manager guide.
- loadbalancer in the *Command Line Interface Reference*
- About TLS-terminated HTTPS load balancers
- Creating a TLS-terminated HTTPS load balancer with SNI
- Creating HTTP and TLS-terminated HTTPS load balancing on the same IP and back end

## 8.5. 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 using a shared IP. It is a best practice to also create a health monitor to ensure that your back end members remain available.

**Prerequisites**

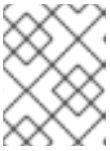
- A private subnet that contains back end servers that host non-secure HTTP applications on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.
- TLS public-key cryptography has been 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.
  - The keys are not encrypted with passphrases.

- You must configure the Load-balancing service (octavia) to use the Key Manager service (barbican). (See link to the *Manage Secrets with OpenStack Key Manager guide* in "Additional resources" later in this topic.)

**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

   The values in parentheses are provided as examples. Replace these with values appropriate for your site.

   ```
   $ 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. Using the Key Manager service, create secret resources (**tls_secret1** and **tls_secret2**) for the PKCS12 file.

   ```
   $ 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)"
   ```

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

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

4. Monitor the state of the load balancer.

   **Example**

   ```
   $ openstack loadbalancer show lb1
   ```

   When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next step.

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

```
$ openstack loadbalancer listener create --protocol-port 443 \
--protocol TERMINATED_HTTPS --name listener1 \
--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.

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

7. Create a health monitor on the pool (**pool1**) that connects to the back end servers and tests the path (/).

   Example

```
$ openstack loadbalancer healthmonitor create --delay 5 --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.

```
$ 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 steps

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   Example

```
$ openstack loadbalancer show lb1
```

   You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| 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          | 2021-01-15T11:12:42                  |
| 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.
   A working member (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   **Example**

   ```
   $ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                |                                      |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   | subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
   | updated_at          | 2021-01-15T11:12:42                  |
   | weight              | 1                                    |
   | monitor_port        | None                                 |
   | monitor_address     | None                                 |
   | backup              | False                                |
   +---------------------+--------------------------------------+
   ```

**Additional resources**

- Manage Secrets with OpenStack Key Manager guide.
- loadbalancer in the *Command Line Interface Reference*
- About TLS-terminated HTTPS load balancers
- Creating a TLS-terminated HTTPS load balancer
- Creating HTTP and TLS-terminated HTTPS load balancing on the same IP and back end

## 8.6. 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 same IP address) when you want to respond to web clients with the exact same content, regardless if the client is connected with 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 private subnet that contains back end servers that host non-secure HTTP applications on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.
- TLS public-key cryptography has been 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 key is not encrypted with a passphrase.
  - The intermediate certificate chain is 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). (See link to the *Manage Secrets with OpenStack Key Manager guide* in "Additional resources" later in this topic.)
- 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

The values in parentheses are provided as examples. Replace these with values appropriate for your site.

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

2. Using the Key Manager service, create a secret resource (**tls_secret1**) for the PKCS12 file.

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

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

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

4. Monitor the state of the load balancer:

   Example

```
$ openstack loadbalancer show lb1
```

   When you see a status of **ACTIVE**, the load balancer is created and running and you can go to the next step.

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

```
$ openstack loadbalancer listener create --protocol-port 443 --protocol
TERMINATED_HTTPS --name listener1 --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.

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

7. Create a health monitor on the pool (**pool1**) that connects to the back end servers and tests the path (/):

   Example

```
$ openstack loadbalancer healthmonitor create --delay 5 --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.

```
$ 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
```

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

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

**Verification steps**

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   Example

   ```
   $ openstack loadbalancer show lb1
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | 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          | 2021-01-15T11:12:42                  |
   | 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.
   A working member (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

   Example

   ```
   $ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | address             | 192.0.2.10                           |
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
   | name                |                                      |
   | operating_status    | ONLINE                               |
   | project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
   | protocol_port       | 80                                   |
   | provisioning_status | ACTIVE                               |
   ```

```
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2021-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

**Additional resources**

- Manage Secrets with OpenStack Key Manager guide.
- loadbalancer in the *Command Line Interface Reference*
- About TLS-terminated HTTPS load balancers
- Creating a TLS-terminated HTTPS load balancer
- Creating a TLS-terminated HTTPS load balancer with SNI

# CHAPTER 9. 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.

## 9.1. CREATING A TCP LOAD BALANCER

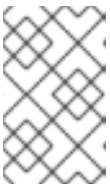
You can use the OpenStack client to create a load balancer when you need to manage network traffic for non-HTTP, TCP-based services and applications. It is a best practice to also create a health monitor to ensure that your back end members remain available.

### Prerequisites

- A private subnet that contains back end servers that host a custom application on a specific TCP port.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.

### Procedure

1. 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 ones that are appropriate for your site.

**Example**

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

2. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running, and you can go to the next step.

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

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

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

Example

```
$ openstack loadbalancer healthmonitor create --delay 5 --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

```
$ 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 steps

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

Example

```
$ openstack loadbalancer show lb1
```

You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| 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          | 2021-01-15T11:12:42                  |
| 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. Obtain a member ID using the following command.

**Example**

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

A working member (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

**Example**

```
$ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
```

You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                |                                      |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 80                                   |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2021-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

**Additional resources**

- loadbalancer in the *Command Line Interface Reference*
- Creating a UDP load balancer with a health monitor
- Creating a QoS-ruled load balancer
- Creating a load balancer with an access control list

## 9.2. CREATING A UDP LOAD BALANCER WITH A HEALTH MONITOR

You can use the OpenStack client to 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 private subnet that contains back end servers that host one or more applications configured to use UDP ports.
- A shared external (public) subnet that is reachable from the internet.
- These back end servers have been configured with a UDP health check.
- Ensure that no security rules block ICMP Destination Unreachable messages (ICMP type 3).

### Procedure

1. 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 ones that are appropriate for your site.

**Example**

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

2. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running, and you can go to the next step.

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**

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

5. Create a health monitor on the pool (**pool1**) that connects to the back end servers using UDP (**UDP-CONNECT**).

   Example

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

   ```
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.10 --
   protocol-port 1234 pool1
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.11 --
   protocol-port 1234 pool1
   ```

Verification steps

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

   Example

   ```
   $ openstack loadbalancer show lb1
   ```

   You should see output similar to the following:

   ```
   +---------------------+--------------------------------------+
   | Field               | Value                                |
   +---------------------+--------------------------------------+
   | admin_state_up      | True                                 |
   | created_at          | 2021-01-15T11:11:09                  |
   | 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          | 2021-01-15T11:12:42                  |
   | 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. Obtain a member ID using the following command.

**Example**

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

A working member (**b85c807e-4d7c-4cbd-b725-5e8afddf80d2**) should have an **ONLINE** value for its **operating_status**.

**Example**

```
$ openstack loadbalancer member show pool1 b85c807e-4d7c-4cbd-b725-5e8afddf80d2
```

You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| address             | 192.0.2.10                           |
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| id                  | b85c807e-4d7c-4cbd-b725-5e8afddf80d2 |
| name                |                                      |
| operating_status    | ONLINE                               |
| project_id          | dda678ca5b1241e7ad7bf7eb211a2fd7     |
| protocol_port       | 1234                                 |
| provisioning_status | ACTIVE                               |
| subnet_id           | 5bd7334b-49b3-4849-b3a2-b0b83852dba1 |
| updated_at          | 2021-01-15T11:12:42                  |
| weight              | 1                                    |
| monitor_port        | None                                 |
| monitor_address     | None                                 |
| backup              | False                                |
+---------------------+--------------------------------------+
```

**Additional resources**

- loadbalancer in the *Command Line Interface Reference*
- Creating a TCP load balancer
- Creating a QoS-ruled load balancer
- Creating a load balancer with an access control list

## 9.3. CREATING A QOS-RULED LOAD BALANCER

You can apply an OpenStack Networking Quality of Service (QoS) policy to virtual IP addresses (VIPs) used by 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.

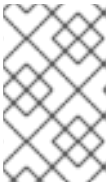
**Prerequisites**

- A private subnet that contains back end servers that have been configured with an HTTP application on TCP port 80.

- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.
- A QoS policy that contains bandwidth limit rules created for OpenStack Networking.

**Procedure**

1. Using the OpenStack client, 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 ones 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
```

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

3. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running, and you can go to the next 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**

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

6. Create a health monitor on the pool that connects to the back end servers and tests the path (/).

   Example

   ```
   $ openstack loadbalancer healthmonitor create --delay 5 --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

   ```
   $ 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 steps

- Run the **openstack loadbalancer list** command to verify the listener ( **listener1**) settings.

  Example

  ```
  $ openstack loadbalancer list
  ```

  You should see output similar to the following:

  ```
  +---------------------+--------------------------------------+
  | Field               | Value                                |
  +---------------------+--------------------------------------+
  | admin_state_up      | True                                 |
  | created_at          | 2021-01-15T11:11:09                  |
  | 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          | 2021-01-15T11:12:42                  |
  | 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*
- network qos policy create in the *Command Line Interface Reference*
- Creating a TCP load balancer
- Creating a UDP load balancer with a health monitor
- Creating a load balancer with an access control list

## 9.4. CREATING A LOAD BALANCER WITH AN ACCESS CONTROL LIST

You can use the OpenStack client to 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 private subnet that contains back end servers that have been configured with a custom application on TCP port 80.
- These back end servers have been configured with a health check at the URL path /.
- A shared external (public) subnet that is reachable from the internet.

### Procedure

1. 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 ones that are appropriate for your site.

**Example**

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

2. Verify the state of the load balancer.

**Example**

```
$ openstack loadbalancer show lb1
```

When you see a status of **ACTIVE**, the load balancer is created and running, and you can go to the next step.

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

   ```
   $ 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 (/).

   Example

   ```
   $ openstack loadbalancer healthmonitor create --delay 5 --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

   ```
   $ 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 steps

1. Run the **openstack loadbalancer listener show** command to verify the listener ( **listener1**) settings.

   Example

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

   You should see output similar to the following:

   ```
   +---------------------------+--------------------------------------+
   | Field                     | Value                                |
   +---------------------------+--------------------------------------+
   | admin_state_up            | True                                 |
   | connection_limit          | -1                                   |
   | created_at                | 2021-01-15T11:11:09                  |
   | 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              | 2021-01-15T11:12:42                |
| 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, try to make a request to the listener from a client whose CIDR is not in the **allowed_cidrs** list; the request should not succeed.
You should see output similar to the following:

```
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*
- Creating a TCP load balancer
- Creating a UDP load balancer with a health monitor
- Creating a QoS-ruled load balancer

## 9.5. 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.

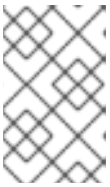
IMPORTANT

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 private subnet that contains back end servers that host a custom application on a specific TCP port.
- A shared external (public) subnet that is reachable from the internet.

**Procedure**

1. 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 ones that are appropriate for your site.

**Example**

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

2. Verify the state of the load balancer:

```
$ openstack loadbalancer show lb1
```

When you see a status of *ACTIVE*, the load balancer is created and running and you can go to the next step.

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**

```
$ 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

```
$ 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 steps

1. Run the **openstack loadbalancer show** command to verify the load balancer ( **lb1**) settings.

Example

```
$ openstack loadbalancer show lb1
```

You should see output similar to the following:

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| created_at          | 2021-01-15T11:11:09                  |
| 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          | 2021-01-15T11:12:42                  |
| 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
```

You should see output similar to the following:

```
+---------------------------+--------------------------------------+
| Field                     | Value                                |
+---------------------------+--------------------------------------+
| admin_state_up            | True                                 |
| connection_limit          | -1                                   |
| created_at                | 2021-01-15T11:13:52                  |
| 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                | 2021-01-15T11:15:17                  |
| 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
```

You should see output similar to the following:

```
+----------------------+--------------------------------------+
| Field                | Value                                |
+----------------------+--------------------------------------+
| admin_state_up       | True                                 |
| created_at           | 2021-01-15T11:17:34                  |
| 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           | 2021-01-15T11:18:59                  |
| tls_container_ref    | None                                 |
| ca_tls_container_ref | None                                 |
| crl_container_ref    | None                                 |
| tls_enabled          | False                                |
+----------------------+--------------------------------------+
```

**Additional resources**

- Load-balancing service (octavia) feature support matrix
- loadbalancer in the *Command Line Interface Reference*
- Load-balancing service provider drivers
- Creating a TCP load balancer
- Creating a UDP load balancer with a health monitor
- Creating a QoS-ruled load balancer
- Creating a load balancer with an access control list

# CHAPTER 10. 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 using several criteria to meet your business needs.

## 10.1. WHAT IS 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 should service any given request. This allows for the application infrastructure to be specifically tuned and optimized to serve specific types of content. For example, one group of back end servers (a pool) can be tuned 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. In fact, it is generally expected that an L7 load balancer expects the back end servers from different pools have different content. L7 load balancers are capable of directing requests based on URI, host, HTTP headers, and other data in the application message.

**Additional resources**

- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.2. LAYER 7 LOAD BALANCING IN THE LOAD-BALANCING SERVICE

While layer 7 (L7) load balancing can be implemented 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.

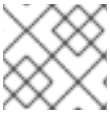
Neutron LBaaS and octavia accomplish the logic of L7 load balancing through the use of L7 rules and policies. An L7 rule is a single, simple logical test which evaluates to true or false. An L7 policy is a collection of L7 rules, as well as a defined action that should be taken if all the rules associated with the policy match.

**Additional resources**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.3. 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 which returns either true or false. It consists of a rule type, a comparison type, a value, and an optional key that gets 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**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic

- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.4. 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 does a comparison between 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 will match 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 will match 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**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.5. 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 do comparisons. The types of comparisons octavia supports are listed below. Note that 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**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.6. LAYER 7 LOAD-BALANCING RULE RESULT INVERSION

To more fully express the logic required by some policies used by the Red Hat OpenStack Platform Load-balancing service (octavia), 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 will be 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**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing policies

- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.7. 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 describe actions that should be taken by the load balancer if all of the rules in the policy are true.

**NOTE**

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

**Additional resources**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.8. LAYER 7 LOAD-BALANCING POLICY LOGIC

For the Red Hat OpenStack Platform Load-balancing service (octavia), layer 7 load-balancing policy logic is very simple: 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, then do this by creating multiple policies with the same action (or, by making a more elaborate regular expression).

**Additional resources**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types

- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy actions
- Layer 7 load-balancing policy position

## 10.9. LAYER 7 LOAD-BALANCING POLICY ACTIONS

If layer 7 load-balancing policy matches a given request, then that policy's action is executed. The following are the actions an L7 policy may 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**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy position

## 10.10. 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. Here are a few notes about how policy position affects listener behavior:

- In the reference implementation of octavia (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 only evaluated after all of the above, and in the order specified by the position of the policy.

- L7 policies are evaluated in a specific order (as defined by the position attribute), and the first policy that matches a given request will be 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 will just be 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, policy C's position becomes **2**.

**Additional resources**

- What is layer 7 load balancing?
- Layer 7 load balancing in octavia
- Layer 7 load-balancing rules
- Layer 7 load-balancing rule types
- Layer 7 load-balancing rule comparison types
- Layer 7 load-balancing rule result inversion
- Layer 7 load-balancing policies
- Layer 7 load-balancing policy logic
- Layer 7 load-balancing policy actions

## 10.11. 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 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 those prerequisites, see the TLS-terminated topic in "Additional resources" later.

### Procedure

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

   #### Example

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

2. 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 --action REDIRECT_PREFIX --redirect-prefix
   https://www.example.com/ --name policy1 http_listener
   ```

3. 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 steps

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

- Creating a TLS-terminated HTTPS load balancer
- loadbalancer in the *Command Line Interface Reference*
- Redirecting requests based on the starting path to a pool
- Sending subdomain requests to a specific pool
- Sending requests based on the host name ending to a specific pool
- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool

- Sending requests to a pool whose name matches the hostname and path
- Setting up A-B testing on an existing production site using a cookie

## 10.12. 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. A layer 7 (L7) policy can be defined 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 those prerequisites, see the HTTP Load balancer topic in "Additional resources" later.

### Procedure

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

   #### Example

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

2. 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

   ```
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.10 --
   protocol-port 80 static_pool
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.11 --
   protocol-port 80 static_pool
   ```

3. 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 --action REDIRECT_TO_POOL --redirect-pool
   static_pool --name policy1 listener1
   ```

4. 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
   ```

5. 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 --action REDIRECT_TO_POOL --redirect-pool
   static_pool --name policy2 listener1
   ```

6. 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 steps**

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**

- Creating an HTTP load balancer with a health monitor
- Redirecting unsecure HTTP requests to secure HTTP
- Sending subdomain requests to a specific pool
- Sending requests based on the host name ending to a specific pool
- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool
- Sending requests to a pool whose name matches the hostname and path
- Setting up A-B testing on an existing production site using a cookie

## 10.13. 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 those prerequisites, see the HTTP Load balancer topic in "Additional resources" later.

### Procedure

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

   #### Example

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

2. 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 --action REDIRECT_TO_POOL --redirect-pool pool2 --name policy1 listener1
   ```

3. 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 steps

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

- Creating an HTTP load balancer with a health monitor
- loadbalancer in the *Command Line Interface Reference*
- Redirecting unsecure HTTP requests to secure HTTP
- Redirecting requests based on the starting path to a pool
- Sending requests based on the host name ending to a specific pool

- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool
- Sending requests to a pool whose name matches the hostname and path
- Setting up A-B testing on an existing production site using a cookie

## 10.14. 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 ending 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 servers, **pool2**.

### Prerequisites

- An HTTP load balancer (**lb1**) that has a listener (**listener1**) and a pool (**pool1**). For those prerequisites, see the HTTP Load balancer topic in "Additional resources" later.

### Procedure

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

   #### Example

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

2. 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 --action REDIRECT_TO_POOL --redirect-pool pool2 --name policy1 listener1
   ```

3. Add an L7 rule to the policy that sends any requests using 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 steps

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
   ```

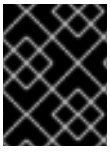
Additional resources

- Creating an HTTP load balancer with a health monitor
- loadbalancer in the *Command Line Interface Reference*
- Redirecting unsecure HTTP requests to secure HTTP
- Redirecting requests based on the starting path to a pool
- Sending subdomain requests to a specific pool
- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool
- Sending requests to a pool whose name matches the hostname and path
- Setting up A-B testing on an existing production site using a cookie

## 10.15. 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 is used to determine 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 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 those prerequisites, see the TLS-terminated topic in "Additional resources" later.
- A second OpenStack Networking subnet on which a secure authentication server authenticates web users.

Procedure

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

Example

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

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

   Example

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

3. 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
   ```

4. 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 steps

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

- Creating a TLS-terminated HTTPS load balancer
- loadbalancer in the *Command Line Interface Reference*
- Redirecting unsecure HTTP requests to secure HTTP
- Redirecting requests based on the starting path to a pool
- Sending subdomain requests to a specific pool

- Sending requests based on the host name ending to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool
- Sending requests to a pool whose name matches the hostname and path
- Setting up A-B testing on an existing production site using a cookie

## 10.16. 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 is used to determine 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 lack the browser cookie, **auth_token**, or have **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 those prerequisites, see the TLS-terminated topic in "Additional resources" later.
- A second OpenStack Networking subnet on which a secure authentication server authenticates web users.

### Procedure

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

   #### Example

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

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

   #### Example

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

3. 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
```

4. 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
   ```

5. 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
   ```

6. 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 steps**

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**

- Creating a TLS-terminated HTTPS load balancer
- loadbalancer in the *Command Line Interface Reference*
- Redirecting unsecure HTTP requests to secure HTTP
- Redirecting requests based on the starting path to a pool
- Sending subdomain requests to a specific pool

- Sending requests based on the host name ending to a specific pool
- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests to a pool whose name matches the hostname and path
- Setting up A-B testing on an existing production site using a cookie

## 10.17. 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 those prerequisites, see the HTTP Load balancer topic in "Additional resources" later.

### Procedure

1. Create a second pool (**api_pool**) on the load balancer (**lb1**).

   #### Example

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

2. 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

   ```
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.10 --
   protocol-port 80 static_pool
   $ openstack loadbalancer member create --subnet-id private_subnet --address 192.0.2.11 --
   protocol-port 80 static_pool
   ```

3. 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
   ```

4. 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
```

5. 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 steps**

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**

- Creating an HTTP load balancer with a health monitor
- loadbalancer in the *Command Line Interface Reference*
- Redirecting unsecure HTTP requests to secure HTTP
- Redirecting requests based on the starting path to a pool
- Sending subdomain requests to a specific pool
- Sending requests based on the host name ending to a specific pool
- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool
- Setting up A-B testing on an existing production site using a cookie

## 10.18. SETTING UP A-B TESTING ON AN EXISTING PRODUCTION SITE USING A COOKIE

You can use the Red Hat OpenStack Platform (RHOSP) Load-balancing service (octavia) with layer 7 (L7) policies to set up A-B testing (or split testing) for your production websites.

In this example, web clients that should be 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).
- An HTTP load balancer has been configured following the instructions for "Redirecting requests based on the starting path to a pool." See the topic link in "Additional resources" later. 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**).

### Procedure

1. Create a third pool (**pool_B**) on a load balancer ( **lb1**).

   #### Example

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

2. 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

   ```
   $ openstack loadbalancer member create --address 192.0.2.50 --protocol-port 80 --subnet-id
   private_subnet pool_B
   $ openstack loadbalancer member create --address 192.0.2.51 --protocol-port 80 --subnet-id
   private_subnet pool_B
   ```

3. Create a fourth pool (**static_pool_B**) on a load balancer ( **lb1**).

   #### Example

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

4. 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 --address 192.0.2.100 --protocol-port 80 --subnet-
   id private_subnet static_pool_B
   $ openstack loadbalancer member create --address 192.0.2.101 --protocol-port 80 --subnet-
   id private_subnet static_pool_B
   ```

5. 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
   ```

6. 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
   ```

7. 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
   ```

8. 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**.

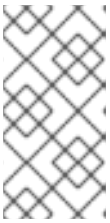
   Example

   ```
   $ openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-pool
   pool_B --name policy3 --position 2 listener1
   ```

9. 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 that L7 policies with the most specific rules be assigned 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 steps

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**

- Creating an HTTP load balancer with a health monitor
- loadbalancer in the *Command Line Interface Reference*
- Redirecting unsecure HTTP requests to secure HTTP
- Redirecting requests based on the starting path to a pool
- Sending subdomain requests to a specific pool
- Sending requests based on the host name ending to a specific pool
- Sending requests based on absence of a browser cookie to a specific pool
- Sending requests based on absence of a browser cookie or invalid cookie value to a specific pool
- Sending requests to a pool whose name matches the hostname and path

# CHAPTER 11. UPDATING AND UPGRADING THE LOAD-BALANCING SERVICE

Performing regular updates and upgrades enables you to leverage the latest Red Hat OpenStack Platform Load-balancing service features and avoid possible lengthy and problematic issues caused by infrequent updates and upgrades.

## 11.1. 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. Stay current with the latest RHOSP updates and upgrades to take advantage of newer load-balancing features.

For Red Hat OpenStack (RHOSP) 13 users who apply the **28 October 2020 maintenance release**, the Load-balancing service is automatically upgraded to the same version that ships in RHOSP 16.1. However, applying the maintenance release must be done only during a scheduled maintenance window. (See "Prerequisites.")

This October maintenance release is a backport in which the octavia container in RHOSP 13 is replaced with the octavia container from RHOSP 16.1, and provides RHOSP 13 users with many of the Load-balancing service features available in RHOSP 16.1.

**Prerequisites**

- Schedule a maintenance window to perform the upgrade, because during the upgrade the octavia control plane is not fully functional.

**Procedure**

1. Perform the RHOSP update as described in the *Keeping Red Hat OpenStack Platform Updated* guide. (For a link to this guide, see "Additional resources," later.)
2. After the maintenance release has been applied, if you need to use new features, then rotate running amphorae to update them to the latest amphora image. (For more information, see the link to "Updating running Load-balancing service instances" in "Additional resources," later.)

**Additional resources**

- Keeping Red Hat OpenStack Platform Updated guide
- Updating running Load-balancing service instances
- Load-balancing service feature support matrix
- Load-balancing service provider drivers

## 11.2. 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 a Red Hat OpenStack Platform (RHOSP) update or upgrade, director automatically downloads the default amphora image, uploads it to the overcloud Image service (glance), and then configures Octavia to use the new image. When you failover a load balancer, you are forcing the Load-balancing service (octavia) to start an instance (amphora) using the new amphora image.

### Prerequisites

- New images for amphora. These are available during an RHOSP update or upgrade.

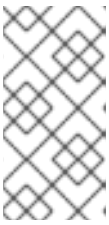
### Procedure

1. List the IDs for all of the load balancers that you want to update:

```
$ openstack loadbalancer list -c id -f value
```

2. 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.

3. 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*
- Updating and upgrading the Load-balancing service

# CHAPTER 12. 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).

## 12.1. VERIFYING THE LOAD BALANCER

One of the basic steps in troubleshooting the Load-balancing service (octavia) and its various components, is to perform the various OpenStack client load balancer show and list commands.

**Procedure**

1. Run the **openstack loadbalancer status show** command to 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 ones that are appropriate for your site.

**Example**

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

2. Run the **loadbalancer amphora list** command to find the UUID of the amphora associated with the load balancer (**lb1**).

**Example**

```
$ openstack loadbalancer amphora list | grep 53a497b3-267d-4abc-968f-94237829f78f
```

3. Run the **loadbalancer amphora show** command with the amphora UUID to view amphora information.

**Example**

```
$ openstack loadbalancer amphora show 62e41d30-1484-4e50-851c-7ab6e16b88d0
```

4. Run the **openstack loadbalancer listener show** command to view the listener ( **listener1**) details.

**Example**

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

5. Run the **openstack loadbalancer pool show** command to view the pool and load-balancer members.

**Example**

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

6. Run the **openstack floating ip list** command to verify the floating IP address.

**Example**

```
$ openstack floating ip list
```

7. Verify HTTPS traffic flows across a load balancer whose listener is configured for **HTTPS** or **TERMINATED_HTTPS** protocols.

**Example**

```
$ curl -v https://198.51.100.213 --insecure
* About to connect() to 198.51.100.213 port 443 (#0)
*   Trying 198.51.100.213...
* Connected to 198.51.100.213 (198.51.100.213) 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: 198.51.100.213
> Accept: */*
>
< HTTP/1.1 200 OK
< Content-Length: 30
<
* Connection #0 to host 198.51.100.213 left intact
```

**Additional resources**

- loadbalancer in the *Command Line Interface Reference*
- Load-balancing service instance administrative logs
- Migrating a specific instance from a host
- Using SSH to connect to load-balancing instances

## 12.2. 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. Tenant flow logs can be sent to and processed by the same syslog receiver used by the administrative logs, but they are configured seperately.

The amphora sends all administrative log messages using the native log format for the application sending the message. The amphorae log to the Red Hat OpenStack Platform (RHOSP) Controller in the same location as the other RHOSP logs (/var/log/containers/octavia/).

### Additional resources

- Managing Load-balancing service instance logs
- Verifying the load balancer
- Migrating a specific instance from a host
- Using SSH to connect to load-balancing instances

## 12.3. MIGRATING A SPECIFIC LOAD-BALANCING SERVICE INSTANCE

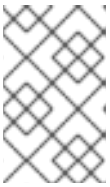
In some cases you must migrate a Load-balancing service instance (amphora) because the host is being shut down for maintenance.

### Procedure

1. 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
```

2. 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 ones that are appropriate for your site.

#### Example

```
$ openstack compute service set compute-host-1 nova-compute --disable
```

3. Fail over the amphora using the amphora ID (**ea17210a-1076-48ff-8a1f-ced49ccb5e53**) that you obtained earlier.

#### 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*
- Verifying the load balancer

- Load-balancing service instance administrative logs
- Using SSH to connect to load-balancing instances

## 12.4. USING SSH TO CONNECT TO LOAD-BALANCING INSTANCES

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 octavia SSH private key.
- You must have SSH enabled in the Load-balancing service configuration prior to creating the load balancer.

### Procedure

1. On the director node (on the undercloud), start the ssh-agent, and add your user identity key to the agent:

```
$ eval `ssh-agent -s`
$ ssh-add
```

2. 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
```

3. Use SSH to connect to the amphora:

```
$ ssh -A -t heat-admin@<controller_node_IP_address> ssh cloud-user@<lb_network_ip>
```

4. When you are finished and have closed you connection to the amphora, stop the SSH agent:

```
$ exit
```

### Additional resources

- loadbalancer in the *Command Line Interface Reference*
- Verifying the load balancer
- Load-balancing service instance administrative logs
- Migrating a specific instance from a host

## 12.5. 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**

- Enter the **loadbalancer listener stats show** command.

  **Example**

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

  **TIP**

  If you do not know the name of the listener, enter the command **loadbalancer listener list**.

**Additional resources**

- loadbalancer listener stats show in the *Command Line Interface Reference*
- Interpreting listener request errors

## 12.6. INTERPRETING LISTENER REQUEST ERRORS

Using the OpenStack Client **loadbalancer listener stats show** command, you can obtain statistics about the listener for a particular Red Hat OpenStack Platform (RHOSP) loadbalancer. (For more information, see Section 12.5, "Showing listener statistics" )

It is important to note that the 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 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*
- Showing listener statistics

# APPENDIX A. LOAD-BALANCING SERVICE COMMAND LINE INTERFACE

The Load-balancing service (octavia) has an OpenStack Client plug-in available as the native Command Line Interface (CLI).

## A.1. LOAD-BALANCING SERVICE COMMAND LINE INTERFACE

See the loadbalancer command in the *Command Line Interface Reference*.