Octavia Documentation

Release 9.1.1.dev69

OpenStack Octavia Team

Mar 06, 2024

©2024 OpenStack Foundation

CONTENTS

1 Octavia Administration 2
- 1.1 Getting Started 2
  - 1.1.1 Introducing Octavia 2
  - 1.1.2 Octavia Glossary 5
  - 1.1.3 Developer / Operator Quick Start Guide 7
- 1.2 Installation and Configuration Guides 14
  - 1.2.1 Building Octavia Amphora Images 14
  - 1.2.2 Octavia Certificate Configuration Guide 21
  - 1.2.3 Octavia Configuration Options 28
  - 1.2.4 Octavia Policies 91
- 1.3 Optional Installation and Configuration Guides 106
  - 1.3.1 Available Provider Drivers 106
  - 1.3.2 Octavia Amphora Log Offloading 108
  - 1.3.3 Octavia API Auditing 112
  - 1.3.4 Octavia API Health Monitoring 116
  - 1.3.5 Octavia Flavors 127
  - 1.3.6 Running Octavia in Apache 129
- 1.4 Maintanence and Operations 129
  - 1.4.1 Operator Maintenance Guide 129
  - 1.4.2 octavia-status 135
  - 1.4.3 Load Balancing Service Upgrade Guide 137
- 1.5 Operator Reference 138
  - 1.5.1 Octavia HAProxy Amphora API 138

2 Octavia Command Line Interface 164

3 Octavia Configuration 165

4 Octavia Contributor 166
- 4.1 Contributor Guidelines 166
  - 4.1.1 So You Want to Contribute... 166
  - 4.1.2 Octavia Constitution 169
  - 4.1.3 Octavia Style Commandments 170
- 4.2 Contributor Reference 173
  - 4.2.1 Provider Driver Development Guide 173
  - 4.2.2 Debugging Octavia code 207
  - 4.2.3 Octavia Entity Relationship Diagram 208
  - 4.2.4 Octavia Controller Flows 208
  - 4.2.5 Guru Meditation Reports 268

- 4.3 Internal APIs . . . 269
- 4.4 Design Documentation . . . 270
  - 4.4.1 Version 0.5 (liberty) . . . 270
- 4.5 Project Specifications . . . 278
  - 4.5.1 Version 0.5 (liberty) . . . 278
  - 4.5.2 Version 0.8 (mitaka) . . . 332
  - 4.5.3 Version 0.9 (newton) . . . 340
  - 4.5.4 Version 1.0 (pike) . . . 366
  - 4.5.5 Version 1.1 (queens) . . . 381
- 4.6 Module Reference . . . 430
  - 4.6.1 octavia . . . 430

5 Octavia Installation 1000

- 5.1 Install and configure . . . 1000
  - 5.1.1 Install and configure for Ubuntu . . . 1000
  - 5.1.2 Additional configuration steps to configure amphorav2 provider . . . 1008

6 Octavia Reference 1011

7 Octavia User 1012

- 7.1 Cookbooks . . . 1012
  - 7.1.1 Basic Load Balancing Cookbook . . . 1012
  - 7.1.2 Layer 7 Cookbook . . . 1032
- 7.2 Guides . . . 1040
  - 7.2.1 Layer 7 Load Balancing . . . 1040
  - 7.2.2 Octavia Provider Feature Matrix . . . 1043
- 7.3 References . . . 1080
  - 7.3.1 Octavia Software Development Kits (SDK) . . . 1080
- 7.4 Videos . . . 1080

Welcome to the OpenStack Octavia project documentation. Octavia brings network load balancing to OpenStack.

See *Introducing Octavia* for an overview of Octavia.

For information on what is new see the Octavia Release Notes.

CHAPTER

ONE

OCTAVIA ADMINISTRATION

1.1 Getting Started

1.1.1 Introducing Octavia

Welcome to Octavia!

Octavia is an open source, operator-scale load balancing solution designed to work with OpenStack.

Octavia was born out of the Neutron LBaaS project. Its conception influenced the transformation of the Neutron LBaaS project, as Neutron LBaaS moved from version 1 to version 2. Starting with the Liberty release of OpenStack, Octavia has become the reference implementation for Neutron LBaaS version 2.

Octavia accomplishes its delivery of load balancing services by managing a fleet of virtual machines, containers, or bare metal serverscollectively known as *amphorae* which it spins up on demand. This on-demand, horizontal scaling feature differentiates Octavia from other load balancing solutions, thereby making Octavia truly suited "for the cloud."

Where Octavia fits into the OpenStack ecosystem

Load balancing is essential for enabling simple or automatic delivery scaling and availability. In turn, application delivery scaling and availability must be considered vital features of any cloud. Together, these facts imply that load balancing is a vital feature of any cloud.

Therefore, we consider Octavia to be as essential as Nova, Neutron, Glance or any other "core" project that enables the essential features of a modern OpenStack cloud.

In accomplishing its role, Octavia makes use of other OpenStack projects:

- **Nova** - For managing amphora lifecycle and spinning up compute resources on demand.
- **Neutron** - For network connectivity between amphorae, tenant environments, and external networks.
- **Barbican** - For managing TLS certificates and credentials, when TLS session termination is configured on the amphorae.
- **Keystone** - For authentication against the Octavia API, and for Octavia to authenticate with other OpenStack projects.
- **Glance** - For storing the amphora virtual machine image.
- **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.

Octavia is designed to interact with the components listed previously. In each case, we've taken care to define this interaction through a driver interface. That way, external components can be swapped out with functionally-equivalent replacements without having to restructure major components of Octavia. For example, if you use an SDN solution other than Neutron in your environment, it should be possible for you to write an Octavia networking driver for your SDN environment, which can be a drop-in replacement for the standard Neutron networking driver in Octavia.

As of Pike, it is recommended to run Octavia as a standalone load balancing solution. Neutron LBaaS is deprecated in the Queens release, and Octavia is its replacement. Whenever possible, operators are **strongly** advised to migrate to Octavia. For end-users, this transition should be relatively seamless, because Octavia supports the Neutron LBaaS v2 API and it has a similar CLI interface. Alternatively, if end-users cannot migrate on their side in the forseable future, operators could enable the experimental Octavia proxy plugin in Neutron LBaaS.

It is also possible to use Octavia as a Neutron LBaaS plugin, in the same way as any other vendor. You can think of Octavia as an "open source vendor" for Neutron LBaaS.

Octavia supports third-party vendor drivers just like Neutron LBaaS, and fully replaces Neutron LBaaS as the load balancing solution for OpenStack.

For further information on OpenStack Neutron LBaaS deprecation, please refer to https://wiki.openstack.org/wiki/Neutron/LBaaS/Deprecation.

Octavia terminology

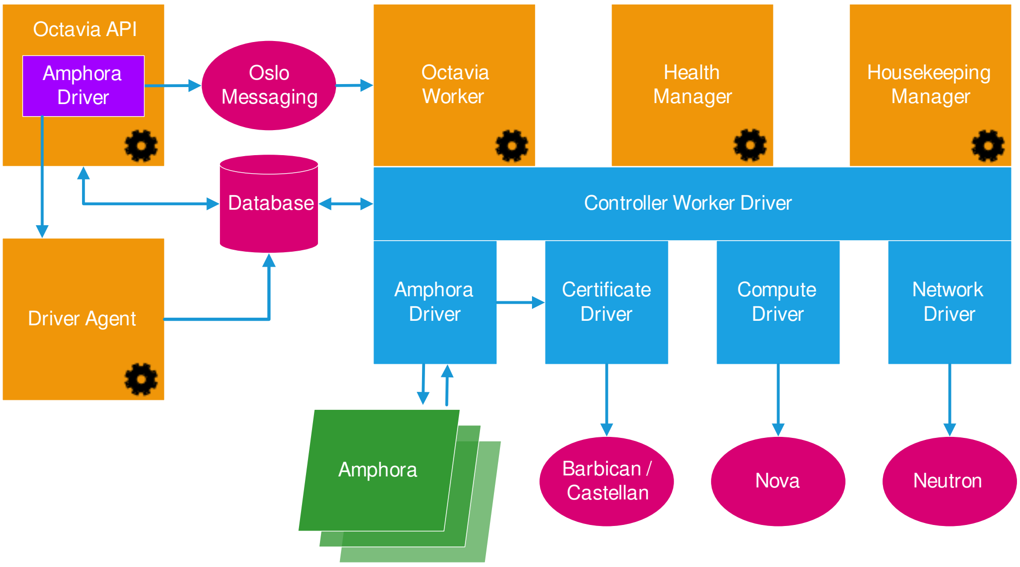
Before you proceed further in this introduction, please note:

Experience shows thatwithin the subsegment of the IT industry that creates, deploys, and uses load balancing devices or services terminology is often used inconsistently. To reduce confusion, the Octavia team has created a glossary of terms as they are defined and used within the context of the Octavia project and Neutron LBaaS version 2. This glossary is available here: *Octavia Glossary*

If you are familiar with Neutron LBaaS version 1 terms and usage, it is especially important for you to understand how the meanings of the terms "VIP," "load balancer," and "load balancing," have changed in Neutron LBaaS version 2.

Our use of these terms should remain consistent with the *Octavia Glossary* throughout Octavia's documentation, in discussions held by Octavia team members on public mailing lists, in IRC channels, and at conferences. To avoid misunderstandings, it's a good idea to familiarize yourself with these glossary definitions.

A 10,000-foot overview of Octavia components


Octavia API
Amphora Driver
Oslo Messaging
Octavia Worker
Health Manager
Housekeeping Manager
Database
Controller Worker Driver
Driver Agent
Amphora Driver
Certificate Driver
Compute Driver
Network Driver
Amphora
Barbican / Castellan
Nova
Neutron

Octavia version 4.0 consists of the following major components:

- **amphorae** - Amphorae are the individual virtual machines, containers, or bare metal servers that accomplish the delivery of load balancing services to tenant application environments. In Octavia version 0.8, the reference implementation of the amphorae image is an Ubuntu virtual machine running HAProxy.
- **controller** - The Controller is the "brains" of Octavia. It consists of five sub-components, which are individual daemons. They can be run on separate back-end infrastructure if desired:
  - **API Controller** - As the name implies, this subcomponent runs Octavia's API. It takes API requests, performs simple sanitizing on them, and ships them off to the controller worker over the Oslo messaging bus.
  - **Controller Worker** - This subcomponent takes sanitized API commands from the API controller and performs the actions necessary to fulfill the API request.
  - **Health Manager** - This subcomponent monitors individual amphorae to ensure they are up and running, and otherwise healthy. It also handles failover events if amphorae fail unexpectedly.
  - **Housekeeping Manager** - This subcomponent cleans up stale (deleted) database records and manages amphora certificate rotation.
  - **Driver Agent** - The driver agent receives status and statistics updates from provider drivers.
- **network** - Octavia cannot accomplish what it does without manipulating the network environment. Amphorae are spun up with a network interface on the "load balancer network," and they may also

plug directly into tenant networks to reach back-end pool members, depending on how any given load balancing service is deployed by the tenant.

For a more complete description of Octavia's components, please see the *Octavia v0.5 Component Design* document within this documentation repository.

1.1.2 Octavia Glossary

As the Octavia project evolves, it's important that people working on Octavia, users using Octavia, and operators deploying Octavia use a common set of terminology in order to avoid misunderstandings and confusion. To that end, we are providing the following glossary of terms.

Note also that many of these terms are expanded upon in design documents in this same repository. What follows is a brief but necessarily incomplete description of these terms.

Amphora Virtual machine, container, dedicated hardware, appliance or device that actually performs the task of load balancing in the Octavia system. More specifically, an amphora takes requests from clients on the front-end and distributes these to back-end systems. Amphorae communicate with their controllers over the LB Network through a driver interface on the controller.

Amphora Load Balancer Driver Component of the controller that does all the communication with amphorae. Drivers communicate with the controller through a generic base class and associated methods, and translate these into control commands appropriate for whatever type of software is running on the back-end amphora corresponding with the driver. This communication happens over the LB network.

Apolocation Term used to describe when two or more amphorae are not colocated on the same physical hardware (which is often essential in HA topologies). May also be used to describe two or more loadbalancers which are not colocated on the same amphora.

Controller Daemon with access to both the LB Network and OpenStack components which coordinates and manages the overall activity of the Octavia load balancing system. Controllers will usually use an abstracted driver interface (usually a base class) for communicating with various other components in the OpenStack environment in order to facilitate loose coupling with these other components. These are the "brains" of the Octavia system.

HAProxy Load balancing software used in the reference implementation for Octavia. (See http://www.haproxy.org/). HAProxy processes run on amphorae and actually accomplish the task of delivering the load balancing service.

Health Monitor An object that defines a check method for each member of the pool. The health monitor itself is a pure-db object which describes the method the load balancing software on the amphora should use to monitor the health of back-end members of the pool with which the health monitor is associated.

L7 Policy

Layer 7 Policy Collection of L7 rules that get logically ANDed together as well as a routing policy for any given HTTP or terminated HTTPS client requests which match said rules. An L7 Policy is associated with exactly one HTTP or terminated HTTPS listener.

For example, a user could specify an L7 policy that any client request that matches the L7 rule "request URI starts with '/api'" should get routed to the "api" pool.

L7 Rule

Layer 7 Rule Single logical expression used to match a condition present in a given HTTP or terminated HTTPS request. L7 rules typically match against a specific header or part of the URI and are used in conjunction with L7 policies to accomplish L7 switching. An L7 rule is associated with exactly one L7 policy.

For example, a user could specify an L7 rule that matches any request URI path that begins with "/api"

L7 Switching

Layer 7 Switching This is a load balancing feature specific to HTTP or terminated HTTPS sessions, in which different client requests are routed to different back-end pools depending on one or more layer 7 policies the user might configure.

For example, using L7 switching, a user could specify that any requests with a URI path that starts with "/api" get routed to the "api" back-end pool, and that all other requests get routed to the default pool.

LB Network Load Balancer Network. The network over which the controller(s) and amphorae communicate. The LB network itself will usually be a nova or neutron network to which both the controller and amphorae have access, but is not associated with any one tenant. The LB Network is generally also *not* part of the undercloud and should not be directly exposed to any OpenStack core components other than the Octavia Controller.

Listener Object representing the listening endpoint of a load balanced service. TCP / UDP port, as well as protocol information and other protocol- specific details are attributes of the listener. Notably, though, the IP address is not.

Load Balancer Object describing a logical grouping of listeners on one or more VIPs and associated with one or more amphorae. (Our "Loadbalancer" most closely resembles a Virtual IP address in other load balancing implementations.) Whether the load balancer exists on more than one amphora depends on the topology used. The load balancer is also often the root object used in various Octavia APIs.

Load Balancing The process of taking client requests on a front-end interface and distributing these to a number of back-end servers according to various rules. Load balancing allows for many servers to participate in delivering some kind TCP or UDP service to clients in an effectively transparent and often highly-available and scalable way (from the client's perspective).

Member Object representing a single back-end server or system that is a part of a pool. A member is associated with only one pool.

Octavia Octavia is an operator-grade open source load balancing solution. Also known as the Octavia system or Octavia project. The term by itself should be used to refer to the system as a whole and not any individual component within the Octavia load balancing system.

Pool Object representing the grouping of members to which the listener forwards client requests. Note that a pool is associated with only one listener, but a listener might refer to several pools (and switch between them using layer 7 policies).

TLS Termination

Transport Layer Security Termination Type of load balancing protocol where HTTPS sessions are terminated (decrypted) on the amphora as opposed to encrypted packets being forwarded on to back-end servers without being decrypted on the amphora. Also known as SSL termination. The main advantages to this type of load balancing are that the payload can be read and / or manipulated by the amphora, and that the expensive tasks of handling the encryption are off-loaded from the back-end servers. This is particularly useful if layer 7 switching is employed in the same listener configuration.

VIP

Virtual IP Address Single service IP address which is associated with a load balancer. This is similar to what is described here: http://en.wikipedia.org/wiki/Virtual_IP_address In a highly available load balancing topology in Octavia, the VIP might be assigned to several amphorae, and a layer-2 protocol like CARP, VRRP, or HSRP (or something unique to the networking infrastructure) might be used to maintain its availability. In layer-3 (routed) topologies, the VIP address might be assigned to an upstream networking device which routes packets to amphorae, which then load balance requests to back-end members.

1.1.3 Developer / Operator Quick Start Guide

This document is intended for developers and operators. For an end-user guide, please see the end-user quick-start guide and cookbook in this documentation repository.

Running Octavia in devstack

tl;dr

- 8GB RAM minimum
- "vmx" or "svm" in `/proc/cpuinfo`
- Ubuntu 18.04 or later
- On that host, copy and run as root: `octavia/devstack/contrib/new-octavia-devstack.sh`

System requirements

Octavia in devstack with a default (non-HA) configuration will deploy one amphora VM per loadbalancer deployed. The current default amphora image also requires at least 1GB of RAM to run effectively. As such it is important that your devstack environment has enough resources dedicated to it to run all its necessary components. For most devstack environments, the limiting resource will be RAM. At the present time, we recommend at least 12GB of RAM for the standard devstack defaults, or 8GB of RAM if cinder and swift are disabled. More is recommended if you also want to run a couple of application server VMs (so that Octavia has something to load balance within your devstack environment).

Also, because the current implementation of Octavia delivers load balancing services using amphorae that run as Nova virtual machines, it is effectively mandatory to enable nested virtualization. The software will work with software emulated CPUs, but be unusably slow. The idea is to make sure the BIOS of the systems you're running your devstack on have virtualization features enabled (Intel VT-x, AMD-V, etc.), and the virtualization software you're using exposes these features to the guest VM (sometimes called nested virtualization). For more information, see: Configure DevStack with KVM-based Nested Virtualization

The devstack environment we recommend should be running Ubuntu Linux 18.04 or later. These instructions may work for other Linux operating systems or environments. However, most people doing development on Octavia are using Ubuntu for their test environment, so you will probably have the easiest time getting your devstack working with that OS.

Deployment

1. Deploy an Ubuntu 18.04 or later Linux host with at least 8GB of RAM. (This can be a VM, but again, make sure you have nested virtualization features enabled in your BIOS and virtualization software.)
2. Copy `devstack/contrib/new-octavia-devstack.sh` from this source repository onto that host.
3. Run new-octavia-devstack.sh as root.
4. Deploy loadbalancers, listeners, etc.

Running Octavia in production

Notes

Disclaimers

This document is not a definitive guide for deploying Octavia in every production environment. There are many ways to deploy Octavia depending on the specifics and limitations of your situation. For example, in our experience, large production environments often have restrictions, hidden "features" or other elements in the network topology which mean the default Neutron networking stack (with which Octavia was designed to operate) must be modified or replaced with a custom networking solution. This may also mean that for your particular environment, you may need to write your own custom networking driver to plug into Octavia. Obviously, instructions for doing this are beyond the scope of this document.

We hope this document provides the cloud operator or distribution creator with a basic understanding of how the Octavia components fit together practically. Through this, it should become more obvious how components of Octavia can be divided or duplicated across physical hardware in a production cloud environment to aid in achieving scalability and resiliency for the Octavia load balancing system.

In the interest of keeping this guide somewhat high-level and avoiding obsolescence or operator/distribution-specific environment assumptions by specifying exact commands that should be run to accomplish the tasks below, we will instead just describe what needs to be done and leave it to the cloud operator or distribution creator to "do the right thing" to accomplish the task for their environment. If you need guidance on specific commands to run to accomplish the tasks described below, we recommend reading through the plugin.sh script in devstack subdirectory of this project. The devstack plugin exercises all the essential components of Octavia in the right order, and this guide will mostly be an elaboration of this process.

Environment Assumptions

The scope of this guide is to provide a basic overview of setting up all the components of Octavia in a production environment, assuming that the default in-tree drivers and components (including a "standard" Neutron install) are going to be used.

For the purposes of this guide, we will therefore assume the following core components have already been set up for your production OpenStack environment:

- Nova
- Neutron

- Glance
- Barbican (if TLS offloading functionality is enabled)
- Keystone
- Rabbit
- MySQL

Production Deployment Walkthrough

Create Octavia User

By default Octavia will use the ’octavia’ user for keystone authentication, and the admin user for interactions with all other services.

You must:

- Create ’octavia’ user.
- Add the ’admin’ role to this user.

Load Balancer Network Configuration

Octavia makes use of an "LB Network" exclusively as a management network that the controller uses to talk to amphorae and vice versa. All the amphorae that Octavia deploys will have interfaces and IP addresses on this network. Therefore, it’s important that the subnet deployed on this network be sufficiently large to allow for the maximum number of amphorae and controllers likely to be deployed throughout the lifespan of the cloud installation.

At the present time, though IPv4 subnets are used by default for the LB Network (for example: 172.16.0.0/12), IPv6 subnets can be used for the LB Network.

The LB Network is isolated from tenant networks on the amphorae by means of network namespaces on the amphorae. Therefore, operators need not be concerned about overlapping subnet ranges with tenant networks.

You must also create a Neutron security group which will be applied to amphorae created on the LB network. It needs to allow amphorae to send UDP heartbeat packets to the health monitor (by default, UDP port 5555), and ingress on the amphora’s API (by default, TCP port 9443). It can also be helpful to allow SSH access to the amphorae from the controller for troubleshooting purposes (ie. TCP port 22), though this is not strictly necessary in production environments.

Amphorae will send periodic health checks to the controller’s health manager. Any firewall protecting the interface on which the health manager listens must allow these packets from amphorae on the LB Network (by default, UDP port 5555).

Finally, you need to add routing or interfaces to this network such that the Octavia controller (which will be described below) is able to communicate with hosts on this network. This also implies you should have some idea where you’re going to run the Octavia controller components.

You must:

- Create the ’lb-mgmt-net’.
- Assign the ’lb-mgmt-net’ to the admin tenant.

- Create a subnet and assign it to the 'lb-mgmt-net'.
- Create neutron security group for amphorae created on the 'lb-mgmt-net'. which allows appropriate access to the amphorae.
- Update firewall rules on the host running the octavia health manager to allow health check messages from amphorae.
- Add appropriate routing to / from the 'lb-mgmt-net' such that egress is allowed, and the controller (to be created later) can talk to hosts on this network.

Create Amphora Image

Octavia deploys amphorae based on a virtual machine disk image. By default we use the OpenStack diskimage-builder project for this. Scripts to accomplish this are within the diskimage-create directory of this repository. In addition to creating the disk image, configure a Nova flavor to use for amphorae, and upload the disk image to glance.

You must:

- Create amphora disk image using OpenStack diskimage-builder.
- Create a Nova flavor for the amphorae.
- Add amphora disk image to glance.
- Tag the above glance disk image with 'amphora'.

Install Octavia Controller Software

This seems somewhat obvious, but the important things to note here are that you should put this somewhere on the network where it will have access to the database (to be initialized below), the oslo messaging system, and the LB network. Octavia uses the standard python setuptools, so installation of the software itself should be straightforward.

Running multiple instances of the individual Octavia controller components on separate physical hosts is recommended in order to provide scalability and availability of the controller software.

The Octavia controller presently consists of several components which may be split across several physical machines. For the 4.0 release of Octavia, the important (and potentially separable) components are the controller worker, housekeeper, health manager and API controller. Please see the component diagrams elsewhere in this repository's documentation for detailed descriptions of each. Please use the following table for hints on which controller components need access to outside resources:

Component	**Resource**		
	LB Network	Database	OSLO messaging
API	No	Yes	Yes
controller worker	Yes	Yes	Yes
health monitor	Yes	Yes	No
housekeeper	Yes	Yes	No

In addition to talking to each other via Oslo messaging, various controller components must also communicate with other OpenStack components, like nova, neutron, barbican, etc. via their APIs.

You must:

- Pick appropriate host(s) to run the Octavia components.
- Install the dependencies for Octavia.
- Install the Octavia software.

Create Octavia Keys and Certificates

Octavia presently allows for one method for the controller to communicate with amphorae: The amphora REST API. Both amphora API and Octavia controller do bi-directional certificate-based authentication in order to authenticate and encrypt communication. You must therefore create appropriate TLS certificates which will be used for key signing, authentication, and encryption. There is a detailed *Octavia Certificate Configuration Guide* to guide you through this process.

Please note that certificates created with this guide may not meet your organization's security policies, since they are self-signed certificates with arbitrary bit lengths, expiration dates, etc. Operators should obviously follow their own security guidelines in creating these certificates.

In addition to the above, it can sometimes be useful for cloud operators to log into running amphorae to troubleshoot problems. The standard method for doing this is to use SSH from the host running the controller worker. In order to do this, you must create an SSH public/private key pair specific to your cloud (for obvious security reasons). You must add this keypair to nova. You must then also update octavia.conf with the keypair name you used when adding it to nova so that amphorae are initialized with it on boot.

See the Troubleshooting Tips section below for an example of how an operator can SSH into an amphora.

You must:

- Create TLS certificates for communicating with the amphorae.
- Create SSH keys for communicating with the amphorae.
- Add the SSH keypair to nova.

Configuring Octavia

Going into all of the specifics of how Octavia can be configured is actually beyond the scope of this document. For full documentation of this, please see the configuration reference: *Octavia Configuration Options*

A configuration template can be found in `etc/octavia.conf` in this repository.

It's also important to note that this configuration file will need to be updated with UUIDs of the LB network, amphora security group, amphora image tag, SSH key path, TLS certificate path, database credentials, etc.

At a minimum, the configuration should specify the following, beyond the defaults. Your specific environment may require more than this:

Section	Configuration parameter
DEFAULT	transport_url
database	connection
certificates	ca_certificate
certificates	ca_private_key
certificates	ca_private_key_passphrase
controller_worker	amp_boot_network_list
controller_worker	amp_flavor_id
controller_worker	amp_image_owner_id
controller_worker	amp_image_tag
controller_worker	amp_secgroup_list
controller_worker	amp_ssh_key_name[1]
controller_worker	amphora_driver
controller_worker	compute_driver
controller_worker	loadbalancer_topology
controller_worker	network_driver
haproxy_amphora	client_cert
haproxy_amphora	server_ca
health_manager	bind_ip
health_manager	controller_ip_port_list
health_manager	heartbeat_key
keystone_authtoken	admin_password
keystone_authtoken	admin_tenant_name
keystone_authtoken	admin_user
keystone_authtoken	www_authenticate_uri
keystone_authtoken	auth_version
oslo_messaging	topic
oslo_messaging_rabbit	rabbit_host
oslo_messaging_rabbit	rabbit_userid
oslo_messaging_rabbit	rabbit_password

You must:

- Create or update `/etc/octavia/octavia.conf` appropriately.

Initialize Octavia Database

This is controlled through alembic migrations under the octavia/db directory in this repository. A tool has been created to aid in the initialization of the octavia database. This should be available under `/usr/local/bin/octavia-db-manage` on the host on which the octavia controller worker is installed. Note that this tool looks at the `/etc/octavia/octavia.conf` file for its database credentials, so initializing the database must happen after Octavia is configured.

It's also important to note here that all of the components of the Octavia controller will need direct access to the database (including the API handler), so you must ensure these components are able to communicate with whichever host is housing your database.

You must:

---

[1] This is technically optional, but extremely useful for troubleshooting.

- Create database credentials for Octavia.
- Add these to the `/etc/octavia/octavia.conf` file.
- Run `/usr/local/bin/octavia-db-manage upgrade head` on the controller worker host to initialize the octavia database.

Launching the Octavia Controller

We recommend using upstart / systemd scripts to ensure the components of the Octavia controller are all started and kept running. It of course doesn't hurt to first start by running these manually to ensure configuration and communication is working between all the components.

You must:

- Make sure each Octavia controller component is started appropriately.

Install Octavia extension in Horizon

This isn't strictly necessary for all cloud installations, however, if yours makes use of the Horizon GUI interface for tenants, it is probably also a good idea to make sure that it is configured with the Octavia extension.

You may:

- Install the octavia GUI extension in Horizon

Test deployment

If all of the above instructions have been followed, it should now be possible to deploy load balancing services using the OpenStack CLI, communicating with the Octavia v2 API.

Example:

```
# openstack loadbalancer create --name lb1 --vip-subnet-id private-subnet
# openstack loadbalancer show lb1
# openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
```

Upon executing the above, log files should indicate that an amphora is deployed to house the load balancer, and that this load balancer is further modified to include a listener. The amphora should be visible to the octavia or admin tenant using the `openstack server list` command, and the listener should respond on the load balancer's IP on port 80 (with an error 503 in this case, since no pool or members have been defined yetbut this is usually enough to see that the Octavia load balancing system is working). For more information on configuring load balancing services as a tenant, please see the end-user quick-start guide and cookbook.

Troubleshooting Tips

The troubleshooting hints in this section are meant primarily for developers or operators troubleshooting underlying Octavia components, rather than end-users or tenants troubleshooting the load balancing service itself.

SSH into Amphorae

If you are using the reference amphora image, it may be helpful to log into running amphorae when troubleshooting service problems. To do this, first discover the `lb_network_ip` address of the amphora you would like to SSH into by looking in the `amphora` table in the octavia database. Then from the host housing the controller worker, run:

```
ssh -i /etc/octavia/.ssh/octavia_ssh_key ubuntu@[lb_network_ip]
```

1.2 Installation and Configuration Guides

1.2.1 Building Octavia Amphora Images

Octavia is an operator-grade reference implementation for Load Balancing as a Service (LBaaS) for OpenStack. The component of Octavia that does the load balancing is known as amphora. Amphora may be a virtual machine, may be a container, or may run on bare metal. Creating images for bare metal amphora installs is outside the scope of this version but may be added in a future release.

Prerequisites

Python pip should be installed as well as the python modules found in the requirements.txt file.

To do so, you can use the following command on Ubuntu:

```
$ # Install python pip
$ sudo apt install python-pip
$ # Eventually create a virtualenv
$ sudo apt install python-virtualenv
$ virtualenv octavia_disk_image_create
$ source octavia_disk_image_create/bin/activate
$ # Install octavia requirements
$ cd octavia/diskimage-create
$ pip install -r requirements.txt
```

Your cache directory should have at least 1GB available, the working directory will need ~1.5GB, and your image destination will need ~500MB

The script will use the version of diskimage-builder installed on your system, or it can be overridden by setting the following environment variables:

```
DIB_REPO_PATH = /<some directory>/diskimage-builder
DIB_ELEMENTS = /<some directory>/diskimage-builder/elements
```

The following packages are required on each platform:

Ubuntu

```
$ sudo apt install qemu-utils git kpartx debootstrap
```

Fedora, CentOS and Red Hat Enterprise Linux

```
$ sudo dnf install qemu-img git e2fsprogs policycoreutils-python-utils
```

Test Prerequisites

The tox image tests require libguestfs-tools 1.24 or newer. Libguestfs allows testing the Amphora image without requiring root privileges. On Ubuntu systems you also need to give read access to the kernels for the user running the tests:

```
$ sudo chmod 0644 /boot/vmlinuz*
```

Usage

This script and associated elements will build Amphora images. Current support is with an Ubuntu base OS and HAProxy. The script can use Fedora as a base OS but these will not initially be tested or supported. As the project progresses and/or the diskimage-builder project adds support for additional base OS options they may become available for Amphora images. This does not mean that they are necessarily supported or tested.

Note: If your cloud has multiple hardware architectures available to nova, remember to set the appropriate hw_architecture property on the image when you load it into glance. For example, when loading an amphora image built for "amd64" you would add "--property hw_architecture='x86_64'" to your "openstack image create" command line.

The script will use environment variables to customize the build beyond the Octavia project defaults, such as adding elements.

The supported and tested image is created by using the diskimage-create.sh defaults (no command line parameters or environment variables set). As the project progresses we may add additional supported configurations.

Command syntax:

```
$ diskimage-create.sh
        [-a i386 | **amd64** | armhf | aarch64 | ppc64le ]
        [-b **haproxy** ]
        [-c **~/.cache/image-create** | <cache directory> ]
        [-d **focal**/**8** | <other release id> ]
        [-e]
        [-f]
        [-g **repository branch** | stable/train | stable/stein | ... ]
        [-h]
```

(continues on next page)

(continued from previous page)

```
        [-i **ubuntu-minimal** | fedora | centos-minimal | rhel ]
        [-k <kernel package name> ]
        [-l <log file> ]
        [-n]
        [-o **amphora-x64-haproxy** | <filename> ]
        [-p]
        [-r <root password> ]
        [-s **2** | <size in GB> ]
        [-t **qcow2** | tar ]
        [-v]
        [-w <working directory> ]
        [-x]
        [-y]

    '-a' is the architecture type for the image (default: amd64)
    '-b' is the backend type (default: haproxy)
    '-c' is the path to the cache directory (default: ~/.cache/image-create)
    '-d' distribution release id (default on ubuntu: focal)
    '-e' enable complete mandatory access control systems when available␣
↪(default: permissive)
    '-f' disable tmpfs for build
    '-g' build the image for a specific OpenStack Git branch (default:␣
↪current repository branch)
    '-h' display help message
    '-i' is the base OS (default: ubuntu-minimal)
    '-k' is the kernel meta package name, currently only for ubuntu-minimal␣
↪base OS (default: linux-image-virtual)
    '-l' is output logfile (default: none)
    '-n' disable sshd (default: enabled)
    '-o' is the output image file name
    '-p' install amphora-agent from distribution packages (default: disabled)"
    '-r' enable the root account in the generated image (default: disabled)
    '-s' is the image size to produce in gigabytes (default: 2)
    '-t' is the image type (default: qcow2)
    '-v' display the script version
    '-w' working directory for image building (default: .)
    '-x' enable tracing for diskimage-builder
    '-y' enable FIPS 140-2 mode in the amphora image
```

Building Images for Alternate Branches

By default, the diskimage-create.sh script will build an amphora image using the Octavia Git branch of the repository. If you need an image for a specific branch, such as "stable/train", you need to specify the "-g" option with the branch name. An example for "stable/train" would be:

```
diskimage-create.sh -g stable/train
```

Advanced Git Branch/Reference Based Images

If you need to build an image from a local repository or with a specific Git reference or branch, you will need to set some environment variables for diskimage-builder.

Note: These advanced settings will override the "-g" diskimage-create.sh setting.

Building From a Local Octavia Repository

Set the DIB_REPOLOCATION_amphora_agent variable to the location of the Git repository containing the amphora agent:

```
export DIB_REPOLOCATION_amphora_agent=/opt/stack/octavia
```

Building With a Specific Git Reference

Set the DIB_REPOREF_amphora_agent variable to point to the Git branch or reference of the amphora agent:

```
export DIB_REPOREF_amphora_agent=refs/changes/40/674140/7
```

See the *Environment Variables* section below for additional information and examples.

Amphora Agent Upper Constraints

You may also need to specify which version of the OpenStack upper-constraints.txt file will be used to build the image. For example, to specify the "stable/train" upper constraints Git branch, set the following environment variable:

```
export DIB_REPOLOCATION_upper_constraints=https://opendev.org/openstack/
↪requirements/raw/branch/stable/train/upper-constraints.txt
```

See Dependency Management for OpenStack Projects for more information.

Environment Variables

These are optional environment variables that can be set to override the script defaults.

DIB_REPOLOCATION_amphora_agent

- Location of the amphora-agent code that will be installed in the image.
- Default: https://opendev.org/openstack/octavia
- Example: /tmp/octavia

DIB_REPOREF_amphora_agent

- The Git reference to checkout for the amphora-agent code inside the image.

- Default: The current branch
- Example: stable/stein
- Example: refs/changes/40/674140/7

DIB_REPOLOCATION_octavia_lib

- Location of the octavia-lib code that will be installed in the image.
- Default: https://opendev.org/openstack/octavia-lib
- Example: /tmp/octavia-lib

DIB_REPOREF_octavia_lib

- The Git reference to checkout for the octavia-lib code inside the image.
- Default: master or stable branch for released OpenStack series installs.
- Example: stable/ussuri
- Example: refs/changes/19/744519/2

DIB_REPOLOCATION_upper_constraints

- Location of the upper-constraints.txt file used for the image.
- Default: The upper-constraints.txt for the current branch
- Example: https://opendev.org/openstack/requirements/raw/branch/master/upper-constraints.txt
- Example: https://opendev.org/openstack/requirements/raw/branch/stable/train/upper-constraints.txt

CLOUD_INIT_DATASOURCES

- Comma separated list of cloud-int datasources
- Default: ConfigDrive
- Options: NoCloud, ConfigDrive, OVF, MAAS, Ec2, <others>
- Reference: https://launchpad.net/cloud-init

DIB_DISTRIBUTION_MIRROR

- URL to a mirror for the base OS selected
- Default: None

DIB_ELEMENTS

- Override the elements used to build the image
- Default: None

DIB_LOCAL_ELEMENTS

- Elements to add to the build (requires DIB_LOCAL_ELEMENTS_PATH be specified)
- Default: None

DIB_LOCAL_ELEMENTS_PATH

- Path to the local elements directory

- Default: None

DIB_REPO_PATH

- Directory containing diskimage-builder
- Default: <directory above OCTAVIA_HOME>/diskimage-builder
- Reference: https://github.com/openstack/diskimage-builder

OCTAVIA_REPO_PATH

- Directory containing octavia
- Default: <directory above the script location>
- Reference: https://github.com/openstack/octavia

Using distribution packages for amphora agent

By default, amphora agent is installed from Octavia Git repository. To use distribution packages, use the "-p" option.

Note this needs a base system image with the required repositories enabled (for example RDO repositories for CentOS/Fedora). One of these variables must be set:

DIB_LOCAL_IMAGE

- Path to the locally downloaded image
- Default: None

DIB_CLOUD_IMAGES

- Directory base URL to download the image from
- Default: depends on the distribution

RHEL specific variables

Building a RHEL-based image requires:

- a Red Hat Enterprise Linux KVM Guest Image, manually download from the Red Hat Customer Portal. Set the DIB_LOCAL_IMAGE variable to point to the file. More details at: <DIB_REPO_PATH>/elements/rhel
- a Red Hat subscription for the matching Red Hat OpenStack Platform repository if you want to install the amphora agent from the official distribution package (requires setting -p option in diskimage-create.sh). Set the needed registration parameters depending on your configuration. More details at: <DIB_REPO_PATH>/elements/rhel-common

Here is an example with Customer Portal registration and OSP 15 repository:

```
$ export DIB_LOCAL_IMAGE='/tmp/rhel-server-8.0-x86_64-kvm.qcow2'

$ export REG_METHOD='portal' REG_REPOS='rhel-8-server-openstack-15-rpms'

$ export REG_USER='<user>' REG_PASSWORD='<password>' REG_AUTO_ATTACH=true
```

This example uses registration via a Satellite (the activation key must enable an OSP repository):

```
$ export DIB_LOCAL_IMAGE='/tmp/rhel-server-8.1-x86_64-kvm.qcow2'

$ export REG_METHOD='satellite' REG_ACTIVATION_KEY="<activation key>"

$ export REG_SAT_URL="<satellite url>" REG_ORG="<satellite org>"
```

Building in a virtualenv with tox

To make use of a virtualenv for Python dependencies you may run `tox`. Note that you may still need to install binary dependencies on the host for the build to succeed.

If you wish to customize your build modify `tox.ini` to pass on relevant environment variables or command line arguments to the `diskimage-create.sh` script.

```
$ tox -e build
```

Container Support

The Docker command line required to import a tar file created with this script is:

```
$ docker import - image:amphora-x64-haproxy < amphora-x64-haproxy.tar
```

References

This documentation and script(s) leverage prior work by the OpenStack TripleO and Sahara teams. Thank you to everyone that worked on them for providing a great foundation for creating Octavia Amphora images.

- https://opendev.org/openstack/diskimage-builder
- https://opendev.org/openstack/tripleo-image-elements
- https://opendev.org/openstack/sahara-image-elements

Copyright

Copyright 2014 Hewlett-Packard Development Company, L.P.

All Rights Reserved.

Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file except in compliance with the License. You may obtain a copy of the License at

- http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

1.2.2 Octavia Certificate Configuration Guide

This document is intended for Octavia administrators setting up certificate authorities for the two-way TLS authentication used in Octavia for command and control of *Amphora*.

This guide does not apply to the configuration of *TERMINATED_TLS* listeners on load balancers. See the Load Balancing Cookbook for instructions on creating *TERMINATED_TLS* listeners.

Two-way TLS Authentication in Octavia

The Octavia controller processes communicate with the Amphora over a TLS connection much like an HTTPS connection to a website. However, Octavia 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 the TLS 1.3 RFC 8446 for the full handshake.

Phase One

When a controller process, such as the Octavia worker process, connects to an Amphora, the Amphora will present its *server* certificate to the controller. The controller will then validate 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 goes into phase two of the two-way TLS authentication.

Phase Two

Once phase one is complete, the controller will present its *client* certificate to the Amphora. The Amphora will then validate the certificate against the *client* CA certificate stored inside the Amphora. If this certificate is successfully validated, the rest of the TLS handshake will continue to establish the secure communication channel between the controller and the Amphora.

Certificate Lifecycles

The *server* certificates are uniquely generated for each amphora by the controller using the *server* certificate authority certificates and keys. These *server* certificates are automatically rotated by the Octavia housekeeping controller process as they near expiration.

The *client* certificates are used for the Octavia controller processes. These are managed by the operator and due to their use on the control plane of the cloud, typically have a long lifetime.

See the Operator Maintenance Guide for more information about the certificate lifecycles.

Creating the Certificate Authorities

As discussed above, this configuration uses two certificate authorities; one for the *server* certificates, and one for the *client* certificates.

Note: Technically Octavia can be run using just one certificate authority by using it to issue certificates for both roles. However, this weakens the security as a *server* certificate from an amphora could be used to impersonate a controller. We recommend you use two certificate authorities for all deployments outside of testing.

For this document we are going to setup simple OpenSSL based certificate authorities. However, any standards compliant certificate authority software can be used to create the required certificates.

1. Create a working directory for the certificate authorities. Make sure to set the proper permissions on this directory such that others cannot access the private keys, random bits, etc. being generated here.

```
$ mkdir certs
$ chmod 700 certs
$ cd certs
```

2. Create the OpenSSL configuration file. This can be shared between the two certificate authorities.

```
$ vi openssl.cnf
```

```
# OpenSSL root CA configuration file.

[ ca ]
# `man ca`
default_ca = CA_default

[ CA_default ]
# Directory and file locations.
dir                 = ./
certs               = $dir/certs
crl_dir             = $dir/crl
new_certs_dir       = $dir/newcerts
database            = $dir/index.txt
serial              = $dir/serial
RANDFILE            = $dir/private/.rand

# The root key and root certificate.
private_key         = $dir/private/ca.key.pem
certificate         = $dir/certs/ca.cert.pem

# For certificate revocation lists.
crlnumber           = $dir/crlnumber
crl                 = $dir/crl/ca.crl.pem
crl_extensions      = crl_ext
default_crl_days    = 30
```

(continues on next page)

(continued from previous page)

```
# SHA-1 is deprecated, so use SHA-2 instead.
default_md        = sha256

name_opt          = ca_default
cert_opt          = ca_default
default_days      = 3650
preserve          = no
policy            = policy_strict

[ policy_strict ]
# The root CA should only sign intermediate certificates that match.
# See the POLICY FORMAT section of `man ca`.
countryName             = match
stateOrProvinceName     = match
organizationName        = match
organizationalUnitName  = optional
commonName              = supplied
emailAddress            = optional

[ req ]
# Options for the `req` tool (`man req`).
default_bits        = 2048
distinguished_name  = req_distinguished_name
string_mask         = utf8only

# SHA-1 is deprecated, so use SHA-2 instead.
default_md          = sha256

# Extension to add when the -x509 option is used.
x509_extensions     = v3_ca

[ req_distinguished_name ]
# See <https://en.wikipedia.org/wiki/Certificate_signing_request>.
countryName                     = Country Name (2 letter code)
stateOrProvinceName             = State or Province Name
localityName                    = Locality Name
0.organizationName              = Organization Name
organizationalUnitName          = Organizational Unit Name
commonName                      = Common Name
emailAddress                    = Email Address

# Optionally, specify some defaults.
countryName_default             = US
stateOrProvinceName_default     = Oregon
localityName_default            =
0.organizationName_default      = OpenStack
organizationalUnitName_default  = Octavia
emailAddress_default            =
```

(continues on next page)

(continued from previous page)

```
commonName_default              = example.org

[ v3_ca ]
# Extensions for a typical CA (`man x509v3_config`).
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid:always,issuer
basicConstraints = critical, CA:true
keyUsage = critical, digitalSignature, cRLSign, keyCertSign

[ usr_cert ]
# Extensions for client certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = client, email
nsComment = "OpenSSL Generated Client Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer
keyUsage = critical, nonRepudiation, digitalSignature, keyEncipherment
extendedKeyUsage = clientAuth, emailProtection

[ server_cert ]
# Extensions for server certificates (`man x509v3_config`).
basicConstraints = CA:FALSE
nsCertType = server
nsComment = "OpenSSL Generated Server Certificate"
subjectKeyIdentifier = hash
authorityKeyIdentifier = keyid,issuer:always
keyUsage = critical, digitalSignature, keyEncipherment
extendedKeyUsage = serverAuth

[ crl_ext ]
# Extension for CRLs (`man x509v3_config`).
authorityKeyIdentifier=keyid:always
```

3. Make any locally required configuration changes to the openssl.cnf. Some settings to consider are:
   - The default certificate lifetime is 10 years.
   - The default bit length is 2048.
4. Make directories for the two certificate authorities.

```
$ mkdir client_ca
$ mkdir server_ca
```

5. Starting with the *server* certificate authority, prepare the CA.

```
$ cd server_ca
$ mkdir certs crl newcerts private
$ chmod 700 private
$ touch index.txt
$ echo 1000 > serial
```

6. Create the *server* CA key.

   - You will need to specify a passphrase to protect the key file.

```
$ openssl genpkey -algorithm RSA -out private/ca.key.pem -aes-128-cbc -
↪pkeyopt rsa_keygen_bits:4096
$ chmod 400 private/ca.key.pem
```

7. Create the *server* CA certificate.

   - You will need to specify the passphrase used in step 6.
   - You will also be asked to provide details for the certificate. These are up to you and should be appropriate for your organization.
   - You may want to mention this is the *server* CA in the common name field.
   - Since this is the CA certificate, you might want to give it a very long lifetime, such as twenty years shown in this example command.

```
$ openssl req -config ../openssl.cnf -key private/ca.key.pem -new -x509 -
↪days 7300 -sha256 -extensions v3_ca -out certs/ca.cert.pem
```

8. Moving to the *client* certificate authority, prepare the CA.

```
$ cd ../client_ca
$ mkdir certs crl csr newcerts private
$ chmod 700 private
$ touch index.txt
$ echo 1000 > serial
```

9. Create the *client* CA key.

   - You will need to specify a passphrase to protect the key file.

```
$ openssl genpkey -algorithm RSA -out private/ca.key.pem -aes-128-cbc -
↪pkeyopt rsa_keygen_bits:4096
$ chmod 400 private/ca.key.pem
```

10. Create the *client* CA certificate.

    - You will need to specify the passphrase used in step 9.
    - You will also be asked to provide details for the certificate. These are up to you and should be appropriate for your organization.
    - You may want to mention this is the *client* CA in the common name field.
    - Since this is the CA certificate, you might want to give it a very long lifetime, such as twenty years shown in this example command.

```
$ openssl req -config ../openssl.cnf -key private/ca.key.pem -new -x509 -
↪days 7300 -sha256 -extensions v3_ca -out certs/ca.cert.pem
```

11. Create a key for the *client* certificate to use.

    - You can create one certificate and key to be used by all of the controllers or you can create a unique certificate and key for each controller.

- You will need to specify a passphrase to protect the key file.

```
$ openssl genpkey -algorithm RSA -out private/ca.key.pem -aes-128-cbc -
↪pkeyopt rsa_keygen_bits:2048
```

12. Create the certificate request for the *client* certificate used on the controllers.

    - You will need to specify the passphrase used in step 11.
    - You will also be asked to provide details for the certificate. These are up to you and should be appropriate for your organization.
    - You must fill in the common name field.
    - You may want to mention this is the *client* certificate in the common name field, or the individual controller information.

```
$ openssl req -config ../openssl.cnf -new -sha256 -key private/client.key.
↪pem -out csr/client.csr.pem
```

13. Sign the *client* certificate request.

    - You will need to specify the CA passphrase used in step 9.
    - Since this certificate is used on the control plane, you might want to give it a very long lifetime, such as twenty years shown in this example command.

```
$ openssl ca -config ../openssl.cnf -extensions usr_cert -days 7300 -
↪notext -md sha256 -in csr/client.csr.pem -out certs/client.cert.pem
```

14. Create a concatenated *client* certificate and key file.

    - You will need to specify the CA passphrase used in step 11.

```
$ openssl rsa -in private/client.key.pem -out private/client.cert-and-key.
↪pem
$ cat certs/client.cert.pem >> private/client.cert-and-key.pem
```

Configuring Octavia

In this section we will configure Octavia to use the certificates and keys created during the *Creating the Certificate Authorities* section.

1. Copy the required files over to your Octavia controllers.

    - Only the Octavia worker, health manager, and housekeeping processes will need access to these files.
    - The first command should return you to the "certs" directory created in step 1 of the *Creating the Certificate Authorities* section.
    - These commands assume you are running the octavia processes under the "octavia" user.
    - Note, some of these steps should be run with "sudo" and are indicated by the "#" prefix.

```
$ cd ..
# mkdir /etc/octavia/certs
# chmod 700 /etc/octavia/certs
# cp server_ca/private/ca.key.pem /etc/octavia/certs/server_ca.key.pem
# chmod 700 /etc/octavia/certs/server_ca.key.pem
# cp server_ca/certs/ca.cert.pem /etc/octavia/certs/server_ca.cert.pem
# cp client_ca/certs/ca.cert.pem /etc/octavia/certs/client_ca.cert.pem
# cp client_ca/private/client.cert-and-key.pem /etc/octavia/certs/client.
↪cert-and-key.pem
# chmod 700 /etc/octavia/certs/client.cert-and-key.pem
# chown -R octavia.octavia /etc/octavia/certs
```

2. Configure the [certificates] section of the octavia.conf file.

   - Only the Octavia worker, health manager, and housekeeping processes will need these settings.
   - The "<server CA passphrase>" should be replaced with the passphrase that was used in step 6 of the *Creating the Certificate Authorities* section.

```
[certificates]
cert_generator = local_cert_generator
ca_certificate = /etc/octavia/certs/server_ca.cert.pem
ca_private_key = /etc/octavia/certs/server_ca.key.pem
ca_private_key_passphrase = <server CA key passphrase>
```

3. Configure the [controller_worker] section of the octavia.conf file.

   - Only the Octavia worker, health manager, and housekeeping processes will need these settings.

```
[controller_worker]
client_ca = /etc/octavia/certs/client_ca.cert.pem
```

4. Configure the [haproxy_amphora] section of the octavia.conf file.

   - Only the Octavia worker, health manager, and housekeeping processes will need these settings.

```
[haproxy_amphora]
client_cert = /etc/octavia/certs/client.cert-and-key.pem
server_ca = /etc/octavia/certs/server_ca.cert.pem
```

5. Start the controller processes.

```
# systemctl start octavia-worker
# systemctl start octavia-healthmanager
# systemctl start octavia-housekeeping
```

1.2.3 Octavia Configuration Options

Table of Contents

- *Octavia Configuration Options*
  - *DEFAULT*
  - *amphora_agent*
  - *api_settings*
  - *audit*
  - *certificates*
  - *compute*
  - *controller_worker*
  - *database*
  - *driver_agent*
  - *glance*
  - *haproxy_amphora*
  - *health_manager*
  - *house_keeping*
  - *keepalived_vrrp*
  - *keystone_authtoken*
  - *networking*
  - *neutron*
  - *nova*
  - *oslo_messaging*
  - *oslo_messaging_amqp*
  - *oslo_messaging_kafka*
  - *oslo_messaging_notifications*
  - *oslo_messaging_rabbit*
  - *quotas*
  - *service_auth*
  - *task_flow*

DEFAULT

`host`

Type hostname

Default `<server-hostname.example.com>`

This option has a sample default set, which means that its actual default value may vary from the one documented above.

The hostname Octavia is running on

`octavia_plugins`

Type string

Default `hot_plug_plugin`

Name of the controller plugin to use

`debug`

Type boolean

Default `False`

Mutable This option can be changed without restarting.

If set to true, the logging level will be set to DEBUG instead of the default INFO level.

`log_config_append`

Type string

Default `<None>`

Mutable This option can be changed without restarting.

The name of a logging configuration file. This file is appended to any existing logging configuration files. For details about logging configuration files, see the Python logging module documentation. Note that when logging configuration files are used then all logging configuration is set in the configuration file and other logging configuration options are ignored (for example, log-date-format).

Table 1: Deprecated Variations

Group	Name
DEFAULT	log-config
DEFAULT	log_config

`log_date_format`

Type string

Default `%Y-%m-%d %H:%M:%S`

Defines the format string for %(asctime)s in log records. Default: the value above . This option is ignored if log_config_append is set.

`log_file`

Type string

Default `<None>`

(Optional) Name of log file to send logging output to. If no default is set, logging will go to stderr as defined by use_stderr. This option is ignored if log_config_append is set.

Table 2: Deprecated Variations

Group	Name
DEFAULT	logfile

log_dir

Type string

Default `<None>`

(Optional) The base directory used for relative log_file paths. This option is ignored if log_config_append is set.

Table 3: Deprecated Variations

Group	Name
DEFAULT	logdir

watch_log_file

Type boolean

Default `False`

Uses logging handler designed to watch file system. When log file is moved or removed this handler will open a new log file with specified path instantaneously. It makes sense only if log_file option is specified and Linux platform is used. This option is ignored if log_config_append is set.

use_syslog

Type boolean

Default `False`

Use syslog for logging. Existing syslog format is DEPRECATED and will be changed later to honor RFC5424. This option is ignored if log_config_append is set.

use_journal

Type boolean

Default `False`

Enable journald for logging. If running in a systemd environment you may wish to enable journal support. Doing so will use the journal native protocol which includes structured metadata in addition to log messages.This option is ignored if log_config_append is set.

syslog_log_facility

Type string

Default `LOG_USER`

Syslog facility to receive log lines. This option is ignored if log_config_append is set.

use_json

Type boolean

Default `False`

Use JSON formatting for logging. This option is ignored if log_config_append is set.

`use_stderr`

Type boolean

Default `False`

Log output to standard error. This option is ignored if log_config_append is set.

`use_eventlog`

Type boolean

Default `False`

Log output to Windows Event Log.

`log_rotate_interval`

Type integer

Default `1`

The amount of time before the log files are rotated. This option is ignored unless log_rotation_type is set to "interval".

`log_rotate_interval_type`

Type string

Default `days`

Valid Values Seconds, Minutes, Hours, Days, Weekday, Midnight

Rotation interval type. The time of the last file change (or the time when the service was started) is used when scheduling the next rotation.

`max_logfile_count`

Type integer

Default `30`

Maximum number of rotated log files.

`max_logfile_size_mb`

Type integer

Default `200`

Log file maximum size in MB. This option is ignored if "log_rotation_type" is not set to "size".

`log_rotation_type`

Type string

Default `none`

Valid Values interval, size, none

Log rotation type.

Possible values

interval Rotate logs at predefined time intervals.

size Rotate logs once they reach a predefined size.

none Do not rotate log files.

`logging_context_format_string`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [%(request_id)s %(user_identity)s] %(instance)s%(message)s`

Format string to use for log messages with context. Used by oslo_log.formatters.ContextFormatter

`logging_default_format_string`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s`

Format string to use for log messages when context is undefined. Used by oslo_log.formatters.ContextFormatter

`logging_debug_format_suffix`

Type string

Default `%(funcName)s %(pathname)s:%(lineno)d`

Additional data to append to log message when logging level for the message is DEBUG. Used by oslo_log.formatters.ContextFormatter

`logging_exception_prefix`

Type string

Default `%(asctime)s.%(msecs)03d %(process)d ERROR %(name)s %(instance)s`

Prefix each line of exception output with this format. Used by oslo_log.formatters.ContextFormatter

`logging_user_identity_format`

Type string

Default `%(user)s %(tenant)s %(domain)s %(user_domain)s %(project_domain)s`

Defines the format string for %(user_identity)s that is used in logging_context_format_string. Used by oslo_log.formatters.ContextFormatter

`default_log_levels`

Type list

Default `['amqp=WARN', 'amqplib=WARN', 'boto=WARN', 'qpid=WARN', 'sqlalchemy=WARN', 'suds=INFO', 'oslo.messaging=INFO',` `'oslo_messaging=INFO', 'iso8601=WARN', 'requests.packages.urllib3.connectionpool=WARN', 'urllib3.connectionpool=WARN', 'websocket=WARN', 'requests.packages.urllib3.util.retry=WARN', 'urllib3.util.retry=WARN', 'keystonemiddleware=WARN', 'routes.middleware=WARN', 'stevedore=WARN', 'taskflow=WARN', 'keystoneauth=WARN', 'oslo.cache=INFO', 'oslo_policy=INFO', 'dogpile.core.dogpile=INFO']`

List of package logging levels in logger=LEVEL pairs. This option is ignored if log_config_append is set.

`publish_errors`

Type boolean

Default `False`

Enables or disables publication of error events.

`instance_format`

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance that is passed with the log message.

`instance_uuid_format`

Type string

Default `"[instance: %(uuid)s] "`

The format for an instance UUID that is passed with the log message.

`rate_limit_interval`

Type integer

Default `0`

Interval, number of seconds, of log rate limiting.

`rate_limit_burst`

Type integer

Default `0`

Maximum number of logged messages per rate_limit_interval.

`rate_limit_except_level`

Type string

Default `CRITICAL`

Log level name used by rate limiting: CRITICAL, ERROR, INFO, WARNING, DEBUG or empty string. Logs with level greater or equal to rate_limit_except_level are not filtered. An empty string means that all levels are filtered.

`fatal_deprecations`

Type boolean

Default `False`

Enables or disables fatal status of deprecations.

`rpc_conn_pool_size`

Type integer

Default `30`

Minimum Value 1

Size of RPC connection pool.

Table 4: Deprecated Variations

Group	Name
DEFAULT	rpc_conn_pool_size

`conn_pool_min_size`

Type integer

Default `2`

The pool size limit for connections expiration policy

`conn_pool_ttl`

Type integer

Default `1200`

The time-to-live in sec of idle connections in the pool

`executor_thread_pool_size`

Type integer

Default `64`

Size of executor thread pool when executor is threading or eventlet.

Table 5: Deprecated Variations

Group	Name
DEFAULT	rpc_thread_pool_size

`rpc_response_timeout`

Type integer

Default `60`

Seconds to wait for a response from a call.

`transport_url`

Type string

Default `rabbit://`

The network address and optional user credentials for connecting to the messaging backend, in URL format. The expected format is:

driver://[user:pass@]host:port[,[userN:passN@]hostN:portN]/virtual_host?query

Example: rabbit://rabbitmq:password@127.0.0.1:5672//

For full details on the fields in the URL see the documentation of oslo_messaging.TransportURL at https://docs.openstack.org/oslo.messaging/latest/reference/transport.html

`control_exchange`

Type string

Default `octavia`

The default exchange under which topics are scoped. May be overridden by an exchange name specified in the transport_url option.

`rpc_ping_enabled`

Type boolean

Default `False`

Add an endpoint to answer to ping calls. Endpoint is named oslo_rpc_server_ping

`log_options`

Type boolean

Default `True`

Mutable This option can be changed without restarting.

Enables or disables logging values of all registered options when starting a service (at DEBUG level).

`graceful_shutdown_timeout`

Type integer

Default `60`

Mutable This option can be changed without restarting.

Specify a timeout after which a gracefully shutdown server will exit. Zero value means endless wait.

amphora_agent

`agent_server_ca`

Type string

Default `/etc/octavia/certs/client_ca.pem`

The ca which signed the client certificates

`agent_server_cert`

Type string

Default `/etc/octavia/certs/server.pem`

The server certificate for the agent server to use

`agent_server_network_dir`

Type string

Default `<None>`

The directory where new network interfaces are located

`agent_server_network_file`

Type string

Default `<None>`

The file where the network interfaces are located. Specifying this will override any value set for agent_server_network_dir.

> **Warning:** This option is deprecated for removal since Xena. Its value may be silently ignored in the future.
>
> **Reason** New amphora interface management does not support single interface file.

`agent_request_read_timeout`

Type integer

Default `180`

The time in seconds to allow a request from the controller to run before terminating the socket.

`agent_tls_protocol`

Type string

Default `TLSv1.2`

Valid Values TLS, TLSv1, TLSv1.1, TLSv1.2, TLS.CLIENT, TLS.SERVER

Minimum TLS protocol for communication with the amphora agent.

`admin_log_targets`

Type list

Default `<None>`

List of log server ip and port pairs for Administrative logs. Additional hosts are backup to the primary server. If none is specified remote logging is disabled. Example 127.0.0.1:10514, 192.168.0.1:10514

`tenant_log_targets`

Type list

Default `<None>`

List of log server ip and port pairs for tenant traffic logs. Additional hosts are backup to the primary server. If none is specified remote logging is disabled. Example 127.0.0.1:10514, 192.168.0.1:10514 `user_log_facility`

Type integer

Default `0`

Minimum Value 0

Maximum Value 7

LOG_LOCAL facility number to use for user traffic logs.

`administrative_log_facility`

Type integer

Default `1`

Minimum Value 0

Maximum Value 7

LOG_LOCAL facility number to use for amphora processes logs.

`log_protocol`

Type string

Default `UDP`

Valid Values TCP, UDP

The log forwarding transport protocol. One of UDP or TCP.

`log_retry_count`

Type integer

Default `5`

The maximum attempts to retry connecting to the logging host.

`log_retry_interval`

Type integer

Default `2`

The time, in seconds, to wait between retries connecting to the logging host.

`log_queue_size`

Type integer

Default `10000`

The queue size (messages) to buffer log messages.

`logging_template_override`

Type string

Default `<None>`

Custom logging configuration template.

`forward_all_logs`

Type boolean

Default `False`

When True, the amphora will forward all of the system logs (except tenant traffic logs) to the admin log target(s). When False, only amphora specific admin logs will be forwarded.

`disable_local_log_storage`

Type boolean

Default `False`

When True, no logs will be written to the amphora filesystem. When False, log files will be written to the local filesystem.

`amphora_id`

Type string

Default `<None>`

The amphora ID.

`amphora_udp_driver`

Type string

Default `keepalived_lvs`

The UDP API backend for amphora agent.

> **Warning:** This option is deprecated for removal since Wallaby. Its value may be silently ignored in the future.
>
> **Reason** amphora-agent will not support any other backend than keepalived_lvs.

api_settings

`bind_host`

Type ip address

Default `127.0.0.1`

The host IP to bind to

`bind_port`

Type port number

Default `9876`

Minimum Value 0

Maximum Value 65535

The port to bind to

`auth_strategy`

Type string

Default `keystone`

Valid Values noauth, keystone, testing

The auth strategy for API requests.

`allow_pagination`

Type boolean

Default `True`

Allow the usage of pagination

`allow_sorting`

Type boolean

Default `True`

Allow the usage of sorting

`allow_filtering`

Type boolean

Default `True`

Allow the usage of filtering

`allow_field_selection`

Type boolean

Default `True`

Allow the usage of field selection

`pagination_max_limit`

Type string

Default `1000`

The maximum number of items returned in a single response. The string 'infinite' or a negative integer value means 'no limit'

`api_base_uri`

Type string

Default `<None>`

Base URI for the API for use in pagination links. This will be autodetected from the request if not overridden here.

`allow_tls_terminated_listeners`

Type boolean

Default `True`

Allow users to create TLS Terminated listeners?

`allow_ping_health_monitors`

Type boolean

Default `True`

Allow users to create PING type Health Monitors?

`enabled_provider_drivers`

Type dict

Default `{'amphora': 'The Octavia Amphora driver.', 'octavia': 'Deprecated alias of the Octavia Amphora driver.'}`

A comma separated list of dictionaries of the enabled provider driver names and descriptions. Must match the driver name in the octavia.api.drivers entrypoint. Example: amphora:The Octavia Amphora driver.,octavia:Deprecated alias of the Octavia Amphora driver.

`default_provider_driver`

Type string

Default `amphora`

Default provider driver.

`udp_connect_min_interval_health_monitor`

Type integer

Default `3`

The minimum health monitor delay interval for the UDP-CONNECT Health Monitor type. A negative integer value means 'no limit'.

`healthcheck_enabled`

Type boolean

Default `False`

When True, the oslo middleware healthcheck endpoint is enabled in the Octavia API.

`healthcheck_refresh_interval`

Type integer

Default `5`

The interval healthcheck plugins should cache results, in seconds.

`default_listener_ciphers`

Type string

Default `TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:`

Default OpenSSL cipher string (colon-separated) for new TLS-enabled listeners.

`default_pool_ciphers`

Type string

Default `TLS_AES_256_GCM_SHA384:TLS_CHACHA20_POLY1305_SHA256:TLS_AES_128_GCM_SHA256:`

Default OpenSSL cipher string (colon-separated) for new TLS-enabled pools.

`tls_cipher_prohibit_list`

Type string

Default `''`

Colon separated list of OpenSSL ciphers. Usage of these ciphers will be blocked.

Table 6: Deprecated Variations

Group	Name
api_settings	tls_cipher_blacklist

default_listener_tls_versions

Type list

Default `['TLSv1.2', 'TLSv1.3']`

List of TLS versions to use for new TLS-enabled listeners.

default_pool_tls_versions

Type list

Default `['TLSv1.2', 'TLSv1.3']`

List of TLS versions to use for new TLS-enabled pools.

minimum_tls_version

Type string

Default `<None>`

Valid Values SSLv3, TLSv1, TLSv1.1, TLSv1.2, TLSv1.3, <None>

Minimum allowed TLS version for listeners and pools.

default_listener_alpn_protocols

Type list

Default `['h2', 'http/1.1', 'http/1.0']`

List of ALPN protocols to use for new TLS-enabled listeners.

default_pool_alpn_protocols

Type list

Default `['h2', 'http/1.1', 'http/1.0']`

List of ALPN protocols to use for new TLS-enabled pools.

audit

enabled

Type boolean

Default `False`

Enable auditing of API requests

audit_map_file

Type string

Default `/etc/octavia/octavia_api_audit_map.conf`

Path to audit map file for octavia-api service. Used only when API audit is enabled.

`ignore_req_list`

Type string

Default `''`

Comma separated list of REST API HTTP methods to be ignored during audit. For example: auditing will not be done on any GET or POST requests if this is set to "GET,POST". It is used only when API audit is enabled.

certificates

`cert_manager`

Type string

Default `barbican_cert_manager`

Name of the cert manager to use

`cert_generator`

Type string

Default `local_cert_generator`

Name of the cert generator to use

`barbican_auth`

Type string

Default `barbican_acl_auth`

Name of the Barbican authentication method to use

`service_name`

Type string

Default `<None>`

The name of the certificate service in the keystone catalog

`endpoint`

Type string

Default `<None>`

A new endpoint to override the endpoint in the keystone catalog.

`region_name`

Type string

Default `<None>`

Region in Identity service catalog to use for communication with the barbican service.

`endpoint_type`

Type string

Default `publicURL`

The endpoint_type to be used for barbican service.

`ca_certificates_file`

Type string

Default `<None>`

CA certificates file path for the key manager service (such as Barbican).

`insecure`

Type boolean

Default `False`

Disable certificate validation on SSL connections

`ca_certificate`

Type string

Default `/etc/ssl/certs/ssl-cert-snakeoil.pem`

Absolute path to the CA Certificate for signing. Defaults to env[OS_OCTAVIA_TLS_CA_CERT].

`ca_private_key`

Type string

Default `/etc/ssl/private/ssl-cert-snakeoil.key`

Absolute path to the Private Key for signing. Defaults to env[OS_OCTAVIA_TLS_CA_KEY].

`ca_private_key_passphrase`

Type string

Default `<None>`

Passphrase for the Private Key. Defaults to env[OS_OCTAVIA_CA_KEY_PASS] or None.

`server_certs_key_passphrase`

Type string

Default `insecure-key-do-not-use-this-key`

Passphrase for encrypting Amphora Certificates and Private Keys. Must be 32, base64(url) compatible, characters long. Defaults to env[TLS_PASS_AMPS_DEFAULT] or insecure-key-do-not-use-this-key

`signing_digest`

Type string

Default `sha256`

Certificate signing digest. Defaults to env[OS_OCTAVIA_CA_SIGNING_DIGEST] or "sha256".

`cert_validity_time`

Type integer

Default `2592000`

The validity time for the Amphora Certificates (in seconds).

compute

`max_retries`

Type integer

Default `15`

The maximum attempts to retry an action with the compute service.

`retry_interval`

Type integer

Default `1`

Seconds to wait before retrying an action with the compute service.

`retry_backoff`

Type integer

Default `1`

The seconds to backoff retry attempts.

`retry_max`

Type integer

Default `10`

The maximum interval in seconds between retry attempts.

controller_worker

`workers`

Type integer

Default `1`

Minimum Value 1

Number of workers for the controller-worker service.

`amp_active_retries`

Type integer

Default `30`

Retry attempts to wait for Amphora to become active

`amp_active_wait_sec`

Type integer

Default `10`

Seconds to wait between checks on whether an Amphora has become active

`amp_flavor_id`

Type string

Default `''`

Nova instance flavor id for the Amphora

`amp_image_tag`

Type string

Default `''`

Glance image tag for the Amphora image to boot. Use this option to be able to update the image without reconfiguring Octavia.

`amp_image_owner_id`

Type string

Default `''`

Restrict glance image selection to a specific owner ID. This is a recommended security setting.

`amp_ssh_key_name`

Type string

Default `''`

Optional SSH keypair name, in nova, that will be used for the authorized_keys inside the amphora.

`amp_boot_network_list`

Type list

Default `''`

List of networks to attach to the Amphorae. All networks defined in the list will be attached to each amphora.

`amp_secgroup_list`

Type list

Default `''`

List of security groups to attach to the Amphora.

`client_ca`

Type string

Default `/etc/octavia/certs/ca_01.pem`

Client CA for the amphora agent to use

`amphora_driver`

Type string

Default `amphora_haproxy_rest_driver`

Name of the amphora driver to use

`compute_driver`

Type string

Default `compute_nova_driver`

Name of the compute driver to use

`network_driver`

Type string

Default `allowed_address_pairs_driver`

Name of the network driver to use

`volume_driver`

Type string

Default `volume_noop_driver`

Valid Values volume_noop_driver, volume_cinder_driver

Name of the volume driver to use

`image_driver`

Type string

Default `image_glance_driver`

Valid Values image_noop_driver, image_glance_driver

Name of the image driver to use

`distributor_driver`

Type string

Default `distributor_noop_driver`

Name of the distributor driver to use

`statistics_drivers`

Type list

Default `['stats_db']`

List of drivers for updating amphora statistics.

Table 7: Deprecated Variations

Group	Name
health_manager	stats_update_driver

`loadbalancer_topology`

Type string

Default `SINGLE`

Valid Values ACTIVE_STANDBY, SINGLE

Mutable This option can be changed without restarting.

Load balancer topology configuration. SINGLE - One amphora per load balancer. ACTIVE_STANDBY - Two amphora per load balancer.

`user_data_config_drive`

Type boolean

Default `False`

If True, build cloud-init user-data that is passed to the config drive on Amphora boot instead of personality files. If False, utilize personality files.

`amphora_delete_retries`

Type integer

Default `5`

Number of times an amphora delete should be retried.

`amphora_delete_retry_interval`

Type integer

Default `5`

Time, in seconds, between amphora delete retries.

`db_commit_retry_attempts`

Type integer

Default `2000`

The number of times the database action will be attempted.

`db_commit_retry_initial_delay`

Type integer

Default `1`

The initial delay before a retry attempt.

`db_commit_retry_backoff`

Type integer

Default `1`

The time to backoff retry attempts.

`db_commit_retry_max`

Type integer

Default `5`

The maximum amount of time to wait between retry attempts.

database

`sqlite_synchronous`

Type boolean

Default `True`

If True, SQLite uses synchronous mode.

Table 8: Deprecated Variations

Group	Name
DEFAULT	sqlite_synchronous

`backend`

Type string

Default `sqlalchemy`

The back end to use for the database.

Table 9: Deprecated Variations

Group	Name
DEFAULT	db_backend

`connection`

Type string

Default `<None>`

The SQLAlchemy connection string to use to connect to the database.

Table 10: Deprecated Variations

Group	Name
DEFAULT	sql_connection
DATABASE	sql_connection
sql	connection

`slave_connection`

Type string

Default `<None>`

The SQLAlchemy connection string to use to connect to the slave database.

`mysql_sql_mode`

Type string

Default `TRADITIONAL`

The SQL mode to be used for MySQL sessions. This option, including the default, overrides any server-set SQL mode. To use whatever SQL mode is set by the server configuration, set this to no value. Example: mysql_sql_mode=

`mysql_enable_ndb`

Type boolean

Default `False`

If True, transparently enables support for handling MySQL Cluster (NDB).

`connection_recycle_time`

Type integer

Default `3600`

Connections which have been present in the connection pool longer than this number of seconds will be replaced with a new one the next time they are checked out from the pool.

`max_pool_size`

Type integer

Default `5`

Maximum number of SQL connections to keep open in a pool. Setting a value of 0 indicates no limit.

`max_retries`

Type integer

Default `10`

Maximum number of database connection retries during startup. Set to -1 to specify an infinite retry count.

Table 11: Deprecated Variations

Group	Name
DEFAULT	sql_max_retries
DATABASE	sql_max_retries

`retry_interval`

Type integer

Default `10`

Interval between retries of opening a SQL connection.

Table 12: Deprecated Variations

Group	Name
DEFAULT	sql_retry_interval
DATABASE	reconnect_interval

`max_overflow`

Type integer

Default `50`

If set, use this value for max_overflow with SQLAlchemy.

Table 13: Deprecated Variations

Group	Name
DEFAULT	sql_max_overflow
DATABASE	sqlalchemy_max_overflow

`connection_debug`

Type integer

Default `0`

Minimum Value 0

Maximum Value 100

Verbosity of SQL debugging information: 0=None, 100=Everything.

Table 14: Deprecated Variations

Group	Name
DEFAULT	sql_connection_debug

`connection_trace`

Type boolean

Default `False`

Add Python stack traces to SQL as comment strings.

Table 15: Deprecated Variations

Group	Name
DEFAULT	sql_connection_trace

`pool_timeout`

Type integer

Default `<None>`

If set, use this value for pool_timeout with SQLAlchemy.

Table 16: Deprecated Variations

Group	Name
DATABASE	sqlalchemy_pool_timeout

`use_db_reconnect`

Type boolean

Default `False`

Enable the experimental use of database reconnect on connection lost.

`db_retry_interval`

Type integer

Default `1`

Seconds between retries of a database transaction.

`db_inc_retry_interval`

Type boolean

Default `True`

If True, increases the interval between retries of a database operation up to db_max_retry_interval.

`db_max_retry_interval`

Type integer

Default `10`

If db_inc_retry_interval is set, the maximum seconds between retries of a database operation.

`db_max_retries`

Type integer

Default `20`

Maximum retries in case of connection error or deadlock error before error is raised. Set to -1 to specify an infinite retry count.

`connection_parameters`

Type string

Default `''`

Optional URL parameters to append onto the connection URL at connect time; specify as param1=value1&param2=value2&...

driver_agent

`status_socket_path`

Type string

Default `/var/run/octavia/status.sock`

Path to the driver status unix domain socket file.

`stats_socket_path`

Type string

Default `/var/run/octavia/stats.sock`

Path to the driver statistics unix domain socket file.

`get_socket_path`

Type string

Default `/var/run/octavia/get.sock`

Path to the driver get unix domain socket file.

`status_request_timeout`

Type integer

Default `5`

Time, in seconds, to wait for a status update request.

`status_max_processes`

Type integer

Default `50`

Maximum number of concurrent processes to use servicing status updates.

`stats_request_timeout`

Type integer

Default `5`

Time, in seconds, to wait for a statistics update request.

`stats_max_processes`

Type integer

Default `50`

Maximum number of concurrent processes to use servicing statistics updates.

`get_request_timeout`

Type integer

Default `5`

Time, in seconds, to wait for a get request.

`get_max_processes`

Type integer

Default `50`

Maximum number of concurrent processes to use servicing get requests.

`max_process_warning_percent`

Type floating point

Default `0.75`

Minimum Value 0.01

Maximum Value 0.99

Percentage of max_processes (both status and stats) in use to start logging warning messages about an overloaded driver-agent.

`provider_agent_shutdown_timeout`

Type integer

Default `60`

The time, in seconds, to wait for provider agents to shutdown after the exit event has been set.

`enabled_provider_agents`

Type list

Default `''`

List of enabled provider agents. The driver-agent will launch these agents at startup.

glance

`service_name`

Type string

Default `<None>`

The name of the glance service in the keystone catalog

`endpoint`

Type string

Default `<None>`

A new endpoint to override the endpoint in the keystone catalog.

`region_name`

Type string

Default `<None>`

Region in Identity service catalog to use for communication with the OpenStack services.

`endpoint_type`

Type string

Default `publicURL`

Endpoint interface in identity service to use

`ca_certificates_file`

Type string

Default `<None>`

CA certificates file path

`insecure`

Type boolean

Default `False`

Disable certificate validation on SSL connections

haproxy_amphora

`base_path`

Type string

Default `/var/lib/octavia`

Base directory for amphora files.

`base_cert_dir`

Type string

Default `/var/lib/octavia/certs`

Base directory for cert storage.

`haproxy_template`

Type string

Default `<None>`

Custom haproxy template.

`connection_logging`

Type boolean

Default `True`

Set this to False to disable connection logging.

`connection_max_retries`

Type integer

Default `120`

Retry threshold for connecting to amphorae.

`connection_retry_interval`

Type integer

Default `5`

Retry timeout between connection attempts in seconds.

`active_connection_max_retries`

Type integer

Default `15`

Retry threshold for connecting to active amphorae.

`active_connection_rety_interval`

Type integer

Default `2`

Retry timeout between connection attempts in seconds for active amphora.

`failover_connection_max_retries`

Type integer

Default `2`

Retry threshold for connecting to an amphora in failover.

`failover_connection_retry_interval`

Type integer

Default `5`

Retry timeout between connection attempts in seconds for amphora in failover.

`build_rate_limit`

Type integer

Default `-1`

Number of amphorae that could be built per controller worker, simultaneously.

`build_active_retries`

Type integer

Default `120`

Retry threshold for waiting for a build slot for an amphorae.

`build_retry_interval`

Type integer

Default `5`

Retry timeout between build attempts in seconds.

`haproxy_stick_size`

Type string

Default `10k`

Size of the HAProxy stick table. Accepts k, m, g suffixes. Example: 10k

`user_log_format`

Type string

Default `{{ project_id }} {{ lb_id }} %f %ci %cp %t %{+Q}r %ST %B %U %[ssl_c_verify] %{+Q}[ssl_c_s_dn] %b %s %Tt %tsc`

Log format string for user flow logging.

`bind_host`

Type ip address

Default `::`

The host IP to bind to

`bind_port`

Type port number

Default `9443`

Minimum Value 0

Maximum Value 65535

The port to bind to

`lb_network_interface`

Type string

Default `o-hm0`

Network interface through which to reach amphora, only required if using IPv6 link local addresses.

`haproxy_cmd`

Type string

Default `/usr/sbin/haproxy`

The full path to haproxy

`respawn_count`

Type integer

Default `2`

The respawn count for haproxy's upstart script

`respawn_interval`

Type integer

Default `2`

The respawn interval for haproxy's upstart script

`rest_request_conn_timeout`

Type floating point

Default `10`

The time in seconds to wait for a REST API to connect.

`rest_request_read_timeout`

Type floating point

Default `60`

The time in seconds to wait for a REST API response.

`timeout_client_data`

Type integer

Default `50000`

Frontend client inactivity timeout.

`timeout_member_connect`

Type integer

Default `5000`

Backend member connection timeout.

`timeout_member_data`

Type integer

Default `50000`

Backend member inactivity timeout.

`timeout_tcp_inspect`

Type integer

Default `0`

Time to wait for TCP packets for content inspection.

`client_cert`

Type string

Default `/etc/octavia/certs/client.pem`

The client certificate to talk to the agent

`server_ca`

Type string

Default `/etc/octavia/certs/server_ca.pem`

The ca which signed the server certificates

`use_upstart`

Type boolean

Default `True`

If False, use sysvinit.

Warning: This option is deprecated for removal. Its value may be silently ignored in the future.

Reason This is now automatically discovered and configured.

`api_db_commit_retry_attempts`

Type integer

Default `15`

The number of times the database action will be attempted.

`api_db_commit_retry_initial_delay`

Type integer

Default `1`

The initial delay before a retry attempt.

`api_db_commit_retry_backoff`

Type integer

Default `1`

The time to backoff retry attempts.

`api_db_commit_retry_max`

Type integer

Default `5`

The maximum amount of time to wait between retry attempts.

`default_connection_limit`

Type integer

Default `50000`

Default connection_limit for listeners, used when setting "-1" or when unsetting connection_limit with the listener API.

health_manager

`bind_ip`

Type ip address

Default `127.0.0.1`

IP address the controller will listen on for heart beats

`bind_port`

Type port number

Default `5555`

Minimum Value 0

Maximum Value 65535

Port number the controller will listen on for heart beats

`failover_threads`

Type integer

Default `10`

Number of threads performing amphora failovers.

`health_update_threads`

Type integer

Default `<None>`

Number of processes for amphora health update.

`stats_update_threads`

Type integer

Default `<None>`

Number of processes for amphora stats update.

`heartbeat_key`

Type string

Default `<None>`

Mutable This option can be changed without restarting.

key used to validate amphora sending the message

`heartbeat_timeout`

Type integer

Default `60`

Interval, in seconds, to wait before failing over an amphora.

`health_check_interval`

Type integer

Default `3`

Sleep time between health checks in seconds.

`sock_rlimit`

Type integer

Default `0`

sets the value of the heartbeat recv buffer

`controller_ip_port_list`

Type list

Default `[]`

Mutable This option can be changed without restarting.

List of controller ip and port pairs for the heartbeat receivers. Example 127.0.0.1:5555, 192.168.0.1:5555

`heartbeat_interval`

Type integer

Default `10`

Mutable This option can be changed without restarting.

Sleep time between sending heartbeats.

`health_update_driver`

Type string

Default `health_db`

Driver for updating amphora health system.

> **Warning:** This option is deprecated for removal since Victoria. Its value may be silently ignored in the future.
>
> **Reason** This driver interface was removed.

house_keeping

`cleanup_interval`

Type integer

Default `30`

DB cleanup interval in seconds

`amphora_expiry_age`

Type integer

Default `604800`

Amphora expiry age in seconds

`load_balancer_expiry_age`

Type integer

Default `604800`

Load balancer expiry age in seconds

`cert_interval`

Type integer

Default `3600`

Certificate check interval in seconds

`cert_expiry_buffer`

Type integer

Default `1209600`

Seconds until certificate expiration

`cert_rotate_threads`

Type integer

Default `10`

Number of threads performing amphora certificate rotation

keepalived_vrrp

`vrrp_advert_int`

Type integer

Default `1`

Amphora role and priority advertisement interval in seconds.

`vrrp_check_interval`

Type integer

Default `5`

VRRP health check script run interval in seconds.

`vrrp_fail_count`

Type integer

Default `2`

Number of successive failures before transition to a fail state.

`vrrp_success_count`

Type integer

Default `2`

Number of consecutive successes before transition to a success state.

`vrrp_garp_refresh_interval`

Type integer

Default `5`

Time in seconds between gratuitous ARP announcements from the MASTER.

`vrrp_garp_refresh_count`

Type integer

Default `2`

Number of gratuitous ARP announcements to make on each refresh interval.

keystone_authtoken

`www_authenticate_uri`

Type string

Default `<None>`

Complete "public" Identity API endpoint. This endpoint should not be an "admin" endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If you're using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint.

Table 17: Deprecated Variations

Group	Name
keystone_authtoken	auth_uri

`auth_uri`

Type string

Default `<None>`

Complete "public" Identity API endpoint. This endpoint should not be an "admin" endpoint, as it should be accessible by all end users. Unauthenticated clients are redirected to this endpoint to authenticate. Although this endpoint should ideally be unversioned, client support in the wild varies. If you're using a versioned v2 endpoint here, then this should *not* be the same endpoint the service user utilizes for validating tokens, because normal end users may not be able to reach that endpoint. This option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

Warning: This option is deprecated for removal since Queens. Its value may be silently ignored in the future.

Reason The auth_uri option is deprecated in favor of www_authenticate_uri and will be removed in the S release.

`auth_version`

Type string

Default `<None>`

API version of the Identity API endpoint.

`interface`

Type string

Default `internal`

Interface to use for the Identity API endpoint. Valid values are "public", "internal" (default) or "admin".

`delay_auth_decision`

Type boolean

Default `False`

Do not handle authorization requests within the middleware, but delegate the authorization decision to downstream WSGI components.

`http_connect_timeout`

Type integer

Default `<None>`

Request timeout value for communicating with Identity API server.

`http_request_max_retries`

Type integer

Default `3`

How many times are we trying to reconnect when communicating with Identity API Server.

`cache`

Type string

Default `<None>`

Request environment key where the Swift cache object is stored. When auth_token middleware is deployed with a Swift cache, use this option to have the middleware share a caching backend with swift. Otherwise, use the `memcached_servers` option instead.

`certfile`

Type string

Default `<None>`

Required if identity server requires client certificate

`keyfile`

Type string

Default `<None>`

Required if identity server requires client certificate

`cafile`

Type string

Default `<None>`

A PEM encoded Certificate Authority to use when verifying HTTPs connections. Defaults to system CAs.

`insecure`

Type boolean

Default `False`

Verify HTTPS connections.

`region_name`

Type string

Default `<None>`

The region in which the identity server can be found.

`memcached_servers`

Type list

Default `<None>`

Optionally specify a list of memcached server(s) to use for caching. If left undefined, tokens will instead be cached in-process.

Table 18: Deprecated Variations

Group	Name
keystone_authtoken	memcache_servers

`token_cache_time`

Type integer

Default `300`

In order to prevent excessive effort spent validating tokens, the middleware caches previously-seen tokens for a configurable duration (in seconds). Set to -1 to disable caching completely.

`memcache_security_strategy`

Type string

Default `None`

Valid Values None, MAC, ENCRYPT

(Optional) If defined, indicate whether token data should be authenticated or authenticated and encrypted. If MAC, token data is authenticated (with HMAC) in the cache. If ENCRYPT, token data is encrypted and authenticated in the cache. If the value is not one of these options or empty, auth_token will raise an exception on initialization.

`memcache_secret_key`

Type string

Default `<None>`

(Optional, mandatory if memcache_security_strategy is defined) This string is used for key derivation.

`memcache_pool_dead_retry`

Type integer

Default `300`

(Optional) Number of seconds memcached server is considered dead before it is tried again.

`memcache_pool_maxsize`

Type integer

Default `10`

(Optional) Maximum total number of open connections to every memcached server.

`memcache_pool_socket_timeout`

Type integer

Default `3`

(Optional) Socket timeout in seconds for communicating with a memcached server.

`memcache_pool_unused_timeout`

Type integer

Default `60`

(Optional) Number of seconds a connection to memcached is held unused in the pool before it is closed.

`memcache_pool_conn_get_timeout`

Type integer

Default `10`

(Optional) Number of seconds that an operation will wait to get a memcached client connection from the pool.

`memcache_use_advanced_pool`

Type boolean

Default `True`

(Optional) Use the advanced (eventlet safe) memcached client pool.

`include_service_catalog`

Type boolean

Default `True`

(Optional) Indicate whether to set the X-Service-Catalog header. If False, middleware will not ask for service catalog on token validation and will not set the X-Service-Catalog header.

`enforce_token_bind`

Type string

Default `permissive`

Used to control the use and type of token binding. Can be set to: "disabled" to not check token binding. "permissive" (default) to validate binding information if the bind type is of a form known to the server and ignore it if not. "strict" like "permissive" but if the bind type is unknown the token will be rejected. "required" any form of token binding is needed to be allowed. Finally the name of a binding method that must be present in tokens.

`service_token_roles`

Type list

Default `['service']`

A choice of roles that must be present in a service token. Service tokens are allowed to request that an expired token can be used and so this check should tightly control that only actual services should be sending this token. Roles here are applied as an ANY check so any role in this list must be present. For backwards compatibility reasons this currently only affects the allow_expired check.

`service_token_roles_required`

Type boolean

Default `False`

For backwards compatibility reasons we must let valid service tokens pass that don't pass the service_token_roles check as valid. Setting this true will become the default in a future release and should be enabled if possible.

`service_type`

Type string

Default `<None>`

The name or type of the service as it appears in the service catalog. This is used to validate tokens that have restricted access rules.

`auth_type`

Type unknown type

Default `<None>`

Authentication type to load

Table 19: Deprecated Variations

Group	Name
keystone_authtoken	auth_plugin

`auth_section`

Type unknown type

Default `<None>`

Config Section from which to load plugin specific options

networking

`max_retries`

Type integer

Default `15`

The maximum attempts to retry an action with the networking service.

`retry_interval`

Type integer

Default `1`

Seconds to wait before retrying an action with the networking service.

`retry_backoff`

Type integer

Default `1`

The seconds to backoff retry attempts.

`retry_max`

Type integer

Default `10`

The maximum interval in seconds between retry attempts.

`port_detach_timeout`

Type integer

Default `300`

Seconds to wait for a port to detach from an amphora.

`allow_vip_network_id`

Type boolean

Default `True`

Can users supply a network_id for their VIP?

`allow_vip_subnet_id`

Type boolean

Default `True`

Can users supply a subnet_id for their VIP?

`allow_vip_port_id`

Type boolean

Default `True`

Can users supply a port_id for their VIP?

`valid_vip_networks`

Type list

Default `<None>`

List of network_ids that are valid for VIP creation. If this field is empty, no validation is performed.

`reserved_ips`

Type list

Default `['169.254.169.254']`

List of IP addresses reserved from being used for member addresses. IPv6 addresses should be in expanded, uppercase form.

`allow_invisible_resource_usage`

Type boolean

Default `False`

When True, users can use network resources they cannot normally see as VIP or member subnets. Making this True may allow users to access resources on subnets they do not normally have access to via neutron RBAC policies.

neutron

`service_name`

Type string

Default `<None>`

The name of the neutron service in the keystone catalog

`endpoint`

Type string

Default `<None>`

A new endpoint to override the endpoint in the keystone catalog.

`region_name`

Type string

Default `<None>`

Region in Identity service catalog to use for communication with the OpenStack services.

`endpoint_type`

Type string

Default `publicURL`

Endpoint interface in identity service to use

`ca_certificates_file`

Type string

Default `<None>`

CA certificates file path

`insecure`

Type boolean

Default `False`

Disable certificate validation on SSL connections

nova

`service_name`

Type string

Default `<None>`

The name of the nova service in the keystone catalog

`endpoint`

Type string

Default `<None>`

A new endpoint to override the endpoint in the keystone catalog.

`region_name`

Type string

Default `<None>`

Region in Identity service catalog to use for communication with the OpenStack services.

`endpoint_type`

Type string

Default `publicURL`

Endpoint interface in identity service to use

`ca_certificates_file`

Type string

Default `<None>`

CA certificates file path

`insecure`

Type boolean

Default `False`

Disable certificate validation on SSL connections

`enable_anti_affinity`

Type boolean

Default `False`

Flag to indicate if nova anti-affinity feature is turned on. This option is only used when creating amphorae in ACTIVE_STANDBY topology.

`anti_affinity_policy`

Type string

Default `anti-affinity`

Valid Values anti-affinity, soft-anti-affinity

Sets the anti-affinity policy for nova

`random_amphora_name_length`

Type integer

Default `0`

If non-zero, generate a random name of the length provided for each amphora, in the format "a[A-Z0-9]*". Otherwise, the default name format will be used: "amphora-{UUID}".

`availability_zone`

Type string

Default `<None>`

Availability zone to use for creating Amphorae

oslo_messaging

`topic`

Type string

Default `<None>`

oslo_messaging_amqp

`container_name`

Type string

Default `<None>`

Name for the AMQP container. must be globally unique. Defaults to a generated UUID

Table 20: Deprecated Variations

Group	Name
amqp1	container_name

`idle_timeout`

Type integer

Default `0`

Timeout for inactive connections (in seconds)

Table 21: Deprecated Variations

Group	Name
amqp1	idle_timeout

`trace`

Type boolean

Default `False`

Debug: dump AMQP frames to stdout

Table 22: Deprecated Variations

Group	Name
amqp1	trace

`ssl`

Type boolean

Default `False`

Attempt to connect via SSL. If no other ssl-related parameters are given, it will use the system's CA-bundle to verify the server's certificate.

ssl_ca_file

Type string

Default `''`

CA certificate PEM file used to verify the server's certificate

Table 23: Deprecated Variations

Group	Name
amqp1	ssl_ca_file

ssl_cert_file

Type string

Default `''`

Self-identifying certificate PEM file for client authentication

Table 24: Deprecated Variations

Group	Name
amqp1	ssl_cert_file

ssl_key_file

Type string

Default `''`

Private key PEM file used to sign ssl_cert_file certificate (optional)

Table 25: Deprecated Variations

Group	Name
amqp1	ssl_key_file

ssl_key_password

Type string

Default `<None>`

Password for decrypting ssl_key_file (if encrypted)

Table 26: Deprecated Variations

Group	Name
amqp1	ssl_key_password

ssl_verify_vhost

Type boolean

Default `False`

By default SSL checks that the name in the server's certificate matches the hostname in the transport_url. In some configurations it may be preferable to use the virtual hostname instead, for example if the server uses the Server Name Indication TLS extension (rfc6066) to provide a certificate per virtual host. Set ssl_verify_vhost to True if the server's SSL certificate uses the virtual host name instead of the DNS name.

`sasl_mechanisms`

Type string

Default `''`

Space separated list of acceptable SASL mechanisms

Table 27: Deprecated Variations

Group	Name
amqp1	sasl_mechanisms

`sasl_config_dir`

Type string

Default `''`

Path to directory that contains the SASL configuration

Table 28: Deprecated Variations

Group	Name
amqp1	sasl_config_dir

`sasl_config_name`

Type string

Default `''`

Name of configuration file (without .conf suffix)

Table 29: Deprecated Variations

Group	Name
amqp1	sasl_config_name

`sasl_default_realm`

Type string

Default `''`

SASL realm to use if no realm present in username

`connection_retry_interval`

Type integer

Default 1

Minimum Value 1

Seconds to pause before attempting to re-connect.

`connection_retry_backoff`

Type integer

Default `2`

Minimum Value 0

Increase the connection_retry_interval by this many seconds after each unsuccessful failover attempt.

`connection_retry_interval_max`

Type integer

Default `30`

Minimum Value 1

Maximum limit for connection_retry_interval + connection_retry_backoff

`link_retry_delay`

Type integer

Default `10`

Minimum Value 1

Time to pause between re-connecting an AMQP 1.0 link that failed due to a recoverable error.

`default_reply_retry`

Type integer

Default `0`

Minimum Value -1

The maximum number of attempts to re-send a reply message which failed due to a recoverable error.

`default_reply_timeout`

Type integer

Default `30`

Minimum Value 5

The deadline for an rpc reply message delivery.

`default_send_timeout`

Type integer

Default `30`

Minimum Value 5

The deadline for an rpc cast or call message delivery. Only used when caller does not provide a timeout expiry.

`default_notify_timeout`

Type integer

Default `30`

Minimum Value 5

The deadline for a sent notification message delivery. Only used when caller does not provide a timeout expiry.

`default_sender_link_timeout`

Type integer

Default `600`

Minimum Value 1

The duration to schedule a purge of idle sender links. Detach link after expiry.

`addressing_mode`

Type string

Default `dynamic`

Indicates the addressing mode used by the driver. Permitted values: 'legacy' - use legacy non-routable addressing 'routable' - use routable addresses 'dynamic' - use legacy addresses if the message bus does not support routing otherwise use routable addressing

`pseudo_vhost`

Type boolean

Default `True`

Enable virtual host support for those message buses that do not natively support virtual hosting (such as qpidd). When set to true the virtual host name will be added to all message bus addresses, effectively creating a private 'subnet' per virtual host. Set to False if the message bus supports virtual hosting using the 'hostname' field in the AMQP 1.0 Open performative as the name of the virtual host.

`server_request_prefix`

Type string

Default `exclusive`

address prefix used when sending to a specific server

Table 30: Deprecated Variations

Group	Name
amqp1	server_request_prefix

`broadcast_prefix`

Type string

Default `broadcast`

address prefix used when broadcasting to all servers

Table 31: Deprecated Variations

Group	Name
amqp1	broadcast_prefix

group_request_prefix

Type string

Default `unicast`

address prefix when sending to any server in group

Table 32: Deprecated Variations

Group	Name
amqp1	group_request_prefix

rpc_address_prefix

Type string

Default `openstack.org/om/rpc`

Address prefix for all generated RPC addresses

notify_address_prefix

Type string

Default `openstack.org/om/notify`

Address prefix for all generated Notification addresses

multicast_address

Type string

Default `multicast`

Appended to the address prefix when sending a fanout message. Used by the message bus to identify fanout messages.

unicast_address

Type string

Default `unicast`

Appended to the address prefix when sending to a particular RPC/Notification server. Used by the message bus to identify messages sent to a single destination.

anycast_address

Type string

Default `anycast`

Appended to the address prefix when sending to a group of consumers. Used by the message bus to identify messages that should be delivered in a round-robin fashion across consumers.

default_notification_exchange

Type string

Default `<None>`

Exchange name used in notification addresses. Exchange name resolution precedence: Target.exchange if set else default_notification_exchange if set else control_exchange if set else 'notify'

`default_rpc_exchange`

Type string

Default `<None>`

Exchange name used in RPC addresses. Exchange name resolution precedence: Target.exchange if set else default_rpc_exchange if set else control_exchange if set else 'rpc'

`reply_link_credit`

Type integer

Default `200`

Minimum Value 1

Window size for incoming RPC Reply messages.

`rpc_server_credit`

Type integer

Default `100`

Minimum Value 1

Window size for incoming RPC Request messages

`notify_server_credit`

Type integer

Default `100`

Minimum Value 1

Window size for incoming Notification messages

`pre_settled`

Type multi-valued

Default `rpc-cast`

Default `rpc-reply`

Send messages of this type pre-settled. Pre-settled messages will not receive acknowledgement from the peer. Note well: pre-settled messages may be silently discarded if the delivery fails. Permitted values: 'rpc-call' - send RPC Calls pre-settled 'rpc-reply'- send RPC Replies pre-settled 'rpc-cast' - Send RPC Casts pre-settled 'notify' - Send Notifications pre-settled

oslo_messaging_kafka

`kafka_max_fetch_bytes`

Type integer

Default `1048576`

Max fetch bytes of Kafka consumer

`kafka_consumer_timeout`

Type floating point

Default `1.0`

Default timeout(s) for Kafka consumers

`pool_size`

Type integer

Default `10`

Pool Size for Kafka Consumers

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

`conn_pool_min_size`

Type integer

Default `2`

The pool size limit for connections expiration policy

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

`conn_pool_ttl`

Type integer

Default `1200`

The time-to-live in sec of idle connections in the pool

> **Warning:** This option is deprecated for removal. Its value may be silently ignored in the future.
>
> **Reason** Driver no longer uses connection pool.

`consumer_group`

Type string

Default `oslo_messaging_consumer`

Group id for Kafka consumer. Consumers in one group will coordinate message consumption

`producer_batch_timeout`

Type floating point

Default `0.0`

Upper bound on the delay for KafkaProducer batching in seconds

`producer_batch_size`

Type integer

Default `16384`

Size of batch for the producer async send

`compression_codec`

Type string

Default `none`

Valid Values none, gzip, snappy, lz4, zstd

The compression codec for all data generated by the producer. If not set, compression will not be used. Note that the allowed values of this depend on the kafka version

`enable_auto_commit`

Type boolean

Default `False`

Enable asynchronous consumer commits

`max_poll_records`

Type integer

Default `500`

The maximum number of records returned in a poll call

`security_protocol`

Type string

Default `PLAINTEXT`

Valid Values PLAINTEXT, SASL_PLAINTEXT, SSL, SASL_SSL

Protocol used to communicate with brokers

`sasl_mechanism`

Type string

Default `PLAIN`

Mechanism when security protocol is SASL

ssl_cafile

Type string

Default `''`

CA certificate PEM file used to verify the server certificate

ssl_client_cert_file

Type string

Default `''`

Client certificate PEM file used for authentication.

ssl_client_key_file

Type string

Default `''`

Client key PEM file used for authentication.

ssl_client_key_password

Type string

Default `''`

Client key password file used for authentication.

oslo_messaging_notifications

driver

Type multi-valued

Default `''`

The Drivers(s) to handle sending notifications. Possible values are messaging, messagingv2, routing, log, test, noop

Table 33: Deprecated Variations

Group	Name
DEFAULT	notification_driver

transport_url

Type string

Default `<None>`

A URL representing the messaging driver to use for notifications. If not set, we fall back to the same configuration used for RPC.

Table 34: Deprecated Variations

Group	Name
DEFAULT	notification_transport_url

`topics`

Type list

Default `['notifications']`

AMQP topic used for OpenStack notifications.

Table 35: Deprecated Variations

Group	Name
rpc_notifier2	topics
DEFAULT	notification_topics

`retry`

Type integer

Default `-1`

The maximum number of attempts to re-send a notification message which failed to be delivered due to a recoverable error. 0 - No retry, -1 - indefinite

oslo_messaging_rabbit

`amqp_durable_queues`

Type boolean

Default `False`

Use durable queues in AMQP.

`amqp_auto_delete`

Type boolean

Default `False`

Auto-delete queues in AMQP.

Table 36: Deprecated Variations

Group	Name
DEFAULT	amqp_auto_delete

`ssl`

Type boolean

Default `False`

Connect over SSL.

Table 37: Deprecated Variations

Group	Name
oslo_messaging_rabbit	rabbit_use_ssl

`ssl_version`

Type string

Default `''`

SSL version to use (valid only if SSL enabled). Valid values are TLSv1 and SSLv23. SSLv2, SSLv3, TLSv1_1, and TLSv1_2 may be available on some distributions.

Table 38: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_version

ssl_key_file

Type string

Default `''`

SSL key file (valid only if SSL enabled).

Table 39: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_keyfile

ssl_cert_file

Type string

Default `''`

SSL cert file (valid only if SSL enabled).

Table 40: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_certfile

ssl_ca_file

Type string

Default `''`

SSL certification authority file (valid only if SSL enabled).

Table 41: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_ssl_ca_certs

heartbeat_in_pthread

Type boolean

Default `True`

Run the health check heartbeat thread through a native python thread by default. If this option is equal to False then the health check heartbeat will inherit the execution model from the parent process. For example if the parent process has monkey patched the stdlib by using eventlet/greenlet then the heartbeat will be run through a green thread.

Warning: This option is deprecated for removal. Its value may be silently ignored in the future.

`kombu_reconnect_delay`

Type floating point

Default `1.0`

How long to wait before reconnecting in response to an AMQP consumer cancel notification.

Table 42: Deprecated Variations

Group	Name
DEFAULT	kombu_reconnect_delay

`kombu_compression`

Type string

Default `<None>`

EXPERIMENTAL: Possible values are: gzip, bz2. If not set compression will not be used. This option may not be available in future versions.

`kombu_missing_consumer_retry_timeout`

Type integer

Default `60`

How long to wait a missing client before abandoning to send it its replies. This value should not be longer than rpc_response_timeout.

Table 43: Deprecated Variations

Group	Name
oslo_messaging_rabbit	kombu_reconnect_timeout

`kombu_failover_strategy`

Type string

Default `round-robin`

Valid Values round-robin, shuffle

Determines how the next RabbitMQ node is chosen in case the one we are currently connected to becomes unavailable. Takes effect only if more than one RabbitMQ node is provided in config.

`rabbit_login_method`

Type string

Default `AMQPLAIN`

Valid Values PLAIN, AMQPLAIN, RABBIT-CR-DEMO

The RabbitMQ login method.

Table 44: Deprecated Variations

Group	Name
DEFAULT	rabbit_login_method

`rabbit_retry_interval`

Type integer

Default `1`

How frequently to retry connecting with RabbitMQ.

`rabbit_retry_backoff`

Type integer

Default `2`

How long to backoff for between retries when connecting to RabbitMQ.

Table 45: Deprecated Variations

Group	Name
DEFAULT	rabbit_retry_backoff

`rabbit_interval_max`

Type integer

Default `30`

Maximum interval of RabbitMQ connection retries. Default is 30 seconds.

`rabbit_ha_queues`

Type boolean

Default `False`

Try to use HA queues in RabbitMQ (x-ha-policy: all). If you change this option, you must wipe the RabbitMQ database. In RabbitMQ 3.0, queue mirroring is no longer controlled by the x-ha-policy argument when declaring a queue. If you just want to make sure that all queues (except those with auto-generated names) are mirrored across all nodes, run: "rabbitmqctl set_policy HA '^(?!amq.).*' '{"ha-mode": "all"}' "

Table 46: Deprecated Variations

Group	Name
DEFAULT	rabbit_ha_queues

`rabbit_transient_queues_ttl`

Type integer

Default `1800`

Minimum Value 1

Positive integer representing duration in seconds for queue TTL (x-expires). Queues which are unused for the duration of the TTL are automatically deleted. The parameter affects only reply and fanout queues.

`rabbit_qos_prefetch_count`

Type integer

Default `0`

Specifies the number of messages to prefetch. Setting to zero allows unlimited messages.

`heartbeat_timeout_threshold`

Type integer

Default `60`

Number of seconds after which the Rabbit broker is considered down if heartbeat's keep-alive fails (0 disables heartbeat).

`heartbeat_rate`

Type integer

Default `2`

How often times during the heartbeat_timeout_threshold we check the heartbeat.

`direct_mandatory_flag`

Type boolean

Default `True`

(DEPRECATED) Enable/Disable the RabbitMQ mandatory flag for direct send. The direct send is used as reply, so the MessageUndeliverable exception is raised in case the client queue does not exist.MessageUndeliverable exception will be used to loop for a timeout to lets a chance to sender to recover.This flag is deprecated and it will not be possible to deactivate this functionality anymore

Warning: This option is deprecated for removal. Its value may be silently ignored in the future.

Reason Mandatory flag no longer deactivable.

`enable_cancel_on_failover`

Type boolean

Default `False`

Enable x-cancel-on-ha-failover flag so that rabbitmq server will cancel and notify consumerswhen queue is down

quotas

`default_load_balancer_quota`

Type integer

Default `-1`

Default per project load balancer quota.

`default_listener_quota`

Type integer

Default `-1`

Default per project listener quota.

`default_member_quota`

Type integer

Default `-1`

Default per project member quota.

`default_pool_quota`

Type integer

Default `-1`

Default per project pool quota.

`default_health_monitor_quota`

Type integer

Default `-1`

Default per project health monitor quota.

`default_l7policy_quota`

Type integer

Default `-1`

Default per project l7policy quota.

`default_l7rule_quota`

Type integer

Default `-1`

Default per project l7rule quota.

service_auth

`auth_url`

Type unknown type

Default `<None>`

Authentication URL

`auth_type`

Type unknown type

Default `<None>`

Authentication type to load

Table 47: Deprecated Variations

Group	Name
service_auth	auth_plugin

`cafile`

Type string

Default `<None>`

PEM encoded Certificate Authority to use when verifying HTTPs connections.

`certfile`

Type string

Default `<None>`

PEM encoded client certificate cert file

`collect_timing`

Type boolean

Default `False`

Collect per-API call timing information.

`default_domain_id`

Type unknown type

Default `<None>`

Optional domain ID to use with v3 and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

`default_domain_name`

Type unknown type

Default `<None>`

Optional domain name to use with v3 API and v2 parameters. It will be used for both the user and project domain in v3 and ignored in v2 authentication.

`domain_id`

Type unknown type

Default `<None>`

Domain ID to scope to

`domain_name`

Type unknown type

Default `<None>`

Domain name to scope to

`insecure`

Type boolean

Default `False`

Verify HTTPS connections.

`keyfile`

Type string

Default `<None>`

PEM encoded client certificate key file

`password`

Type unknown type

Default `<None>`

User's password

`project_domain_id`

Type unknown type

Default `<None>`

Domain ID containing project

`project_domain_name`

Type unknown type

Default `<None>`

Domain name containing project

`project_id`

Type unknown type

Default `<None>`

Project ID to scope to

Table 48: Deprecated Variations

Group	Name
service_auth	tenant-id
service_auth	tenant_id

`project_name`

Type unknown type

Default `<None>`

Project name to scope to

Table 49: Deprecated Variations

Group	Name
service_auth	tenant-name
service_auth	tenant_name

`split_loggers`

Type boolean

Default `False`

Log requests to multiple loggers.

`system_scope`

Type unknown type

Default `<None>`

Scope for system operations

`tenant_id`

Type unknown type

Default `<None>`

Tenant ID

`tenant_name`

Type unknown type

Default `<None>`

Tenant Name

`timeout`

Type integer

Default `<None>`

Timeout value for http requests

`trust_id`

Type unknown type

Default `<None>`

Trust ID

`user_domain_id`

Type unknown type

Default `<None>`

User's domain id

`user_domain_name`

Type unknown type

Default `<None>`

User's domain name

`user_id`

Type unknown type

Default `<None>`

User id

`username`

Type unknown type

Default `<None>`

Username

Table 50: Deprecated Variations

Group	Name
service_auth	user-name
service_auth	user_name

task_flow

`engine`

Type string

Default `parallel`

Valid Values serial, parallel

TaskFlow engine to use. serial - Runs all tasks on a single thread. parallel - Schedules tasks onto different threads to allow for running non-dependent tasks simultaneously

`max_workers`

Type integer

Default `5`

The maximum number of workers

`disable_revert`

Type boolean

Default `False`

If True, disables the controller worker taskflow flows from reverting. This will leave resources in an inconsistent state and should only be used for debugging purposes.

`persistence_connection`

Type string

Default `sqlite://`

Persistence database, which will be used to store tasks states. Database connection url with db name

`jobboard_enabled`

Type boolean

Default `False`

If True, enables TaskFlow jobboard.

`jobboard_backend_driver`

Type string

Default `redis_taskflow_driver`

Valid Values redis_taskflow_driver, zookeeper_taskflow_driver

Jobboard backend driver that will monitor job state.

`jobboard_backend_hosts`

Type list

Default `['127.0.0.1']`

Jobboard backend server host(s).

`jobboard_backend_port`

Type port number

Default `6379`

Minimum Value 0

Maximum Value 65535

Jobboard backend server port

`jobboard_backend_password`

Type string

Default `''`

Jobboard backend server password

`jobboard_backend_namespace`

Type string

Default `octavia_jobboard`

Jobboard name that should be used to store taskflow job id and claims for it.

`jobboard_redis_sentinel`

Type string

Default `<None>`

Sentinel name if it is used for Redis.

`jobboard_redis_backend_ssl_options`

Type dict

Default `{'ssl': False, 'ssl_keyfile': None, 'ssl_certfile': None, 'ssl_ca_certs': None, 'ssl_cert_reqs': 'required'}`

Redis jobboard backend ssl configuration options.

`jobboard_zookeeper_ssl_options`

Type dict

Default `{'use_ssl': False, 'keyfile': None, 'keyfile_password': None, 'certfile': None, 'verify_certs': True}`

Zookeeper jobboard backend ssl configuration options.

`jobboard_expiration_time`

Type integer

Default `30`

For backends like redis claiming jobs requiring setting the expiry - how many seconds the claim should be retained for.

`jobboard_save_logbook`

Type boolean

Default `False`

If for analysis required saving logbooks info, set this parameter to True. By default remove logbook from persistence backend when job completed.

1.2.4 Octavia Policies

> **Warning:** JSON formatted policy file is deprecated since Octavia 8.0.0 (Wallaby). This oslopolicy-convert-json-to-yaml tool will migrate your existing JSON-formatted policy file to YAML in a backward-compatible way.

Octavia Advanced Role Based Access Control (RBAC)

Octavia adopted the "Advanced Role Based Access Control (RBAC)" default policies in the Pike release of OpenStack. This provides a fine-grained default access control policy for the Octavia service.

The Octavia Advanced RBAC goes beyond the OpenStack legacy RBAC policies of allowing "owners and admins" full access to all services. It also provides a more fine-grained RBAC policy than the newer Keystone Default Roles .

The default policy is to not allow access unless the auth_strategy is 'noauth'.

Users must be a member of one of the following roles to have access to the load-balancer API:

role:load-balancer_observer User has access to load-balancer read-only APIs.

role:load-balancer_global_observer User has access to load-balancer read-only APIs including resources owned by others.

role:load-balancer_member User has access to load-balancer read and write APIs.

role:load-balancer_quota_admin User is considered an admin for quota APIs only.

role:load-balancer_admin User is considered an admin for all load-balancer APIs including resources owned by others.

role:admin and system_scope:all User is admin to all service APIs, including Octavia.

Note: 'is_admin:True' is a policy rule that takes into account the auth_strategy == noauth configuration setting. It is equivalent to 'rule:context_is_admin or {auth_strategy == noauth}' if that would be valid syntax.

These roles are in addition to the Keystone Default Roles:

- role:reader
- role:member

In addition, the Octavia API supports Keystone scoped tokens. When enabled in Oslo Policy, users will need to present a token scoped to either the "system" or a specific "project". See the section *Upgrade Considerations* for more information.

See the section *Managing Octavia User Roles* for examples and advice on how to apply these RBAC policies in production.

Legacy Admin or Owner Policy Override File

An alternate policy file has been provided in octavia/etc/policy called admin_or_owner-policy.yaml that removes the load-balancer RBAC role requirement. Please see the README.rst in that directory for more information.

This will drop the role requirements to allow access to all with the "admin" role or if the user is a member of the project that created the resource. All users have access to the Octavia API to create and manage load balancers under their project.

OpenStack Default Roles Policy Override File

An alternate policy file has been provided in octavia/etc/policy called keystone_default_roles-policy.yaml that removes the load-balancer RBAC role requirement. Please see the README.rst in that directory for more information.

This policy will honor the following Keystone Default Roles in the Octavia API:

- Admin
- Project scoped - Reader
- Project scoped - Member

In addition, there is an alternate policy file that enables system scoped tokens checking called keystone_default_roles_scoped-policy.yaml.

- System scoped - Admin
- System scoped - Reader
- Project scoped - Reader
- Project scoped - Member

Managing Octavia User Roles

User and group roles are managed through the Keystone (identity) project.

A role can be added to a user with the following command:

```
openstack role add --project <project name or id> --user <user name or id>
↪<role>
```

An example where user "jane", in the "engineering" project, gets a new role "load-balancer_member":

```
openstack role add --project engineering --user jane load-balancer_member
```

Keystone Group Roles

Roles can also be assigned to Keystone groups. This can simplify the management of user roles greatly.

For example, your cloud may have a "users" group defined in Keystone. This group is set up to have all of the regular users of your cloud as a member. If you want all of your users to have access to the load balancing service Octavia, you could add the "load-balancer_member" role to the "users" group:

```
openstack role add --domain default --group users load-balancer_member
```

Upgrade Considerations

Starting with the Wallaby release of Octavia, Keystone token scopes and default roles can be enforced. By default, in the Wallaby release, Oslo Policy will not be enforcing these new roles and scopes. However, at some point in the future they may become the default. You may want to enable them now to be ready for the later transition. This section will describe those settings.

The Oslo Policy project defines two configuration settings, among others, that can be set in the Octavia configuration file to influence how policies are handled in the Octavia API. Those two settings are enforce_scope and enforce_new_defaults.

[oslo_policy] enforce_scope

Keystone has introduced the concept of token scopes. Currently, Oslo Policy defaults to not enforce the scope validation of a token for backward compatibility reasons.

The Octavia API supports enforcing the Keystone token scopes as of the Wallaby release. If you are ready to start enforcing the Keystone token scope in the Octavia API you can add the following setting to your Octavia API configuration file:

```
[oslo_policy]
enforce_scope = True
```

Currently the primary effect of this setting is to allow a system scoped admin token when performing administrative API calls to the Octavia API. It will also allow system scoped reader tokens to have the equivalent of the load-balancer_global_observer role.

The Octavia API already enforces the project scoping in Keystone tokens.

[oslo_policy] enforce_new_defaults

The Octavia Wallaby release added support for Keystone Default Roles in the default policies. The previous Octavia Advanced RBAC policies have now been deprecated in favor of the new policies requiring one of the new Keystone Default Roles. Currently, Oslo Policy defaults to using the deprecated policies that do not require the new Keystone Default Roles for backward compatibility.

The Octavia API supports requiring these new Keystone Default Roles as of the Wallaby release. If you are ready to start requiring these roles you can enable the new policies by adding the following setting to your Octavia API configuration file:

```
[oslo_policy]
enforce_new_defaults = True
```

When the new default policies are enabled in the Octavia API, users with the load-balancer:observer role will also require the Keystone default role of "role:reader". Users with the load-balancer:member role will also require the Keystone default role of "role:member".

Sample File Generation

To generate a sample policy.yaml file from the Octavia defaults, run the oslo policy generation script:

```
oslopolicy-sample-generator
--config-file etc/policy/octavia-policy-generator.conf
--output-file policy.yaml.sample
```

Merged File Generation

This will output a policy file which includes all registered policy defaults and all policies configured with a policy file. This file shows the effective policy in use by the project:

```
oslopolicy-policy-generator
--config-file etc/policy/octavia-policy-generator.conf
```

This tool uses the output_file path from the config-file.

List Redundant Configurations

This will output a list of matches for policy rules that are defined in a configuration file where the rule does not differ from a registered default rule. These are rules that can be removed from the policy file with no change in effective policy:

```
oslopolicy-list-redundant
--config-file etc/policy/octavia-policy-generator.conf
```

Default Octavia Policies - API Effective Rules

This section will list the RBAC rules the Octavia API will use followed by a list of the roles that will be allowed access.

Without enforce_scope and enforce_new_defaults:

- load-balancer:read
  - load-balancer_admin
  - load-balancer_global_observer
  - load-balancer_member and <project member>
  - load-balancer_observer and <project member>
  - role:admin
- load-balancer:read-global
  - load-balancer_admin
  - load-balancer_global_observer
  - role:admin
- load-balancer:write

  - load-balancer_admin
  - load-balancer_member and <project member>
  - role:admin
- load-balancer:read-quota
  - load-balancer_admin
  - load-balancer_global_observer
  - load-balancer_member and <project member>
  - load-balancer_observer and <project member>
  - load-balancer_quota_admin
  - role:admin
- load-balancer:read-quota-global
  - load-balancer_admin
  - load-balancer_global_observer
  - load-balancer_quota_admin
  - role:admin
- load-balancer:write-quota
  - load-balancer_admin
  - load-balancer_quota_admin
  - role:admin

With enforce_scope and enforce_new_defaults:

- load-balancer:read
  - load-balancer_admin
  - load-balancer_global_observer
  - load-balancer_member and <project member> and role:member
  - load-balancer_observer and <project member> and role:reader
  - role:admin and system_scope:all
  - role:reader and system_scope:all
- load-balancer:read-global
  - load-balancer_admin
  - load-balancer_global_observer
  - role:admin and system_scope:all
  - role:reader and system_scope:all
- load-balancer:write
  - load-balancer_admin

  - load-balancer_member and <project member> and role:member
  - role:admin and system_scope:all
- load-balancer:read-quota
  - load-balancer_admin
  - load-balancer_global_observer
  - load-balancer_member and <project member> and role:member
  - load-balancer_observer and <project member> and role:reader
  - load-balancer_quota_admin
  - role:admin and system_scope:all
  - role:reader and system_scope:all
- load-balancer:read-quota-global
  - load-balancer_admin
  - load-balancer_global_observer
  - load-balancer_quota_admin
  - role:admin and system_scope:all
  - role:reader and system_scope:all
- load-balancer:write-quota
  - load-balancer_admin
  - load-balancer_quota_admin
  - role:admin and system_scope:all

Default Octavia Policies - Generated From The Octavia Code

```
# Intended scope(s): system
#"system-admin": "role:admin and system_scope:all"

# Intended scope(s): system
#"system-reader": "role:reader and system_scope:all"

# Intended scope(s): project
#"project-member": "role:member and project_id:%(project_id)s"

# Intended scope(s): project
#"project-reader": "role:reader and project_id:%(project_id)s"

# Intended scope(s): system
#"context_is_admin": "role:load-balancer_admin or rule:system-admin"

# DEPRECATED
# "context_is_admin":"role:admin or role:load-balancer_admin" has been
```

(continues on next page)

(continued from previous page)

```
# deprecated since W in favor of "context_is_admin":"role:load-
# balancer_admin or rule:system-admin".
# The Octavia API now requires the OpenStack default roles and scoped
# tokens. See
# https://docs.openstack.org/octavia/latest/configuration/policy.html
# and https://docs.openstack.org/keystone/latest/contributor/services.
# html#reusable-default-roles for more information.

# Intended scope(s): project
#"load-balancer:owner": "project_id:%(project_id)s"

# Intended scope(s): project
#"load-balancer:observer_and_owner": "role:load-balancer_observer and␣
↪rule:project-reader"

# DEPRECATED
# "load-balancer:observer_and_owner":"role:load-balancer_observer and
# rule:load-balancer:owner" has been deprecated since W in favor of
# "load-balancer:observer_and_owner":"role:load-balancer_observer and
# rule:project-reader".
# The Octavia API now requires the OpenStack default roles and scoped
# tokens. See
# https://docs.openstack.org/octavia/latest/configuration/policy.html
# and https://docs.openstack.org/keystone/latest/contributor/services.
# html#reusable-default-roles for more information.

# Intended scope(s): system
#"load-balancer:global_observer": "role:load-balancer_global_observer or␣
↪rule:system-reader"

# Intended scope(s): project
#"load-balancer:member_and_owner": "role:load-balancer_member and␣
↪rule:project-member"

# DEPRECATED
# "load-balancer:member_and_owner":"role:load-balancer_member and
# rule:load-balancer:owner" has been deprecated since W in favor of
# "load-balancer:member_and_owner":"role:load-balancer_member and
# rule:project-member".
# The Octavia API now requires the OpenStack default roles and scoped
# tokens. See
# https://docs.openstack.org/octavia/latest/configuration/policy.html
# and https://docs.openstack.org/keystone/latest/contributor/services.
# html#reusable-default-roles for more information.

# Intended scope(s): system
#"load-balancer:admin": "is_admin:True or role:load-balancer_admin or␣
↪rule:system-admin"
```

(continues on next page)

(continued from previous page)

```
# Intended scope(s): project, system
#"load-balancer:read": "rule:load-balancer:observer_and_owner or rule:load-
↪balancer:global_observer or rule:load-balancer:member_and_owner or␣
↪rule:load-balancer:admin"

# Intended scope(s): system
#"load-balancer:read-global": "rule:load-balancer:global_observer or␣
↪rule:load-balancer:admin"

# Intended scope(s): project, system
#"load-balancer:write": "rule:load-balancer:member_and_owner or rule:load-
↪balancer:admin"

# Intended scope(s): project, system
#"load-balancer:read-quota": "rule:load-balancer:observer_and_owner or␣
↪rule:load-balancer:global_observer or rule:load-balancer:member_and_owner␣
↪or role:load-balancer_quota_admin or rule:load-balancer:admin"

# Intended scope(s): system
#"load-balancer:read-quota-global": "rule:load-balancer:global_observer or␣
↪role:load-balancer_quota_admin or rule:load-balancer:admin"

# Intended scope(s): system
#"load-balancer:write-quota": "role:load-balancer_quota_admin or rule:load-
↪balancer:admin"

# List Flavors
# GET  /v2.0/lbaas/flavors
#"os_load-balancer_api:flavor:get_all": "rule:load-balancer:read"

# Create a Flavor
# POST  /v2.0/lbaas/flavors
#"os_load-balancer_api:flavor:post": "rule:load-balancer:admin"

# Update a Flavor
# PUT  /v2.0/lbaas/flavors/{flavor_id}
#"os_load-balancer_api:flavor:put": "rule:load-balancer:admin"

# Show Flavor details
# GET  /v2.0/lbaas/flavors/{flavor_id}
#"os_load-balancer_api:flavor:get_one": "rule:load-balancer:read"

# Remove a Flavor
# DELETE  /v2.0/lbaas/flavors/{flavor_id}
#"os_load-balancer_api:flavor:delete": "rule:load-balancer:admin"

# List Flavor Profiles
# GET  /v2.0/lbaas/flavorprofiles
#"os_load-balancer_api:flavor-profile:get_all": "rule:load-balancer:admin"
```

(continues on next page)

(continued from previous page)

```
# Create a Flavor Profile
# POST  /v2.0/lbaas/flavorprofiles
#"os_load-balancer_api:flavor-profile:post": "rule:load-balancer:admin"

# Update a Flavor Profile
# PUT  /v2.0/lbaas/flavorprofiles/{flavor_profile_id}
#"os_load-balancer_api:flavor-profile:put": "rule:load-balancer:admin"

# Show Flavor Profile details
# GET  /v2.0/lbaas/flavorprofiles/{flavor_profile_id}
#"os_load-balancer_api:flavor-profile:get_one": "rule:load-balancer:admin"

# Remove a Flavor Profile
# DELETE  /v2.0/lbaas/flavorprofiles/{flavor_profile_id}
#"os_load-balancer_api:flavor-profile:delete": "rule:load-balancer:admin"

# List Availability Zones
# GET  /v2.0/lbaas/availabilityzones
#"os_load-balancer_api:availability-zone:get_all": "rule:load-balancer:read"

# Create an Availability Zone
# POST  /v2.0/lbaas/availabilityzones
#"os_load-balancer_api:availability-zone:post": "rule:load-balancer:admin"

# Update an Availability Zone
# PUT  /v2.0/lbaas/availabilityzones/{availability_zone_id}
#"os_load-balancer_api:availability-zone:put": "rule:load-balancer:admin"

# Show Availability Zone details
# GET  /v2.0/lbaas/availabilityzones/{availability_zone_id}
#"os_load-balancer_api:availability-zone:get_one": "rule:load-balancer:read"

# Remove an Availability Zone
# DELETE  /v2.0/lbaas/availabilityzones/{availability_zone_id}
#"os_load-balancer_api:availability-zone:delete": "rule:load-balancer:admin"

# List Availability Zones
# GET  /v2.0/lbaas/availabilityzoneprofiles
#"os_load-balancer_api:availability-zone-profile:get_all": "rule:load-
↪balancer:admin"

# Create an Availability Zone
# POST  /v2.0/lbaas/availabilityzoneprofiles
#"os_load-balancer_api:availability-zone-profile:post": "rule:load-
↪balancer:admin"

# Update an Availability Zone
# PUT  /v2.0/lbaas/availabilityzoneprofiles/{availability_zone_profile_id}
```

(continues on next page)

(continued from previous page)

```
#"os_load-balancer_api:availability-zone-profile:put": "rule:load-
↪balancer:admin"

# Show Availability Zone details
# GET  /v2.0/lbaas/availabilityzoneprofiles/{availability_zone_profile_id}
#"os_load-balancer_api:availability-zone-profile:get_one": "rule:load-
↪balancer:admin"

# Remove an Availability Zone
# DELETE  /v2.0/lbaas/availabilityzoneprofiles/{availability_zone_profile_id}
#"os_load-balancer_api:availability-zone-profile:delete": "rule:load-
↪balancer:admin"

# List Health Monitors of a Pool
# GET  /v2/lbaas/healthmonitors
#"os_load-balancer_api:healthmonitor:get_all": "rule:load-balancer:read"

# List Health Monitors including resources owned by others
# GET  /v2/lbaas/healthmonitors
#"os_load-balancer_api:healthmonitor:get_all-global": "rule:load-
↪balancer:read-global"

# Create a Health Monitor
# POST  /v2/lbaas/healthmonitors
#"os_load-balancer_api:healthmonitor:post": "rule:load-balancer:write"

# Show Health Monitor details
# GET  /v2/lbaas/healthmonitors/{healthmonitor_id}
#"os_load-balancer_api:healthmonitor:get_one": "rule:load-balancer:read"

# Update a Health Monitor
# PUT  /v2/lbaas/healthmonitors/{healthmonitor_id}
#"os_load-balancer_api:healthmonitor:put": "rule:load-balancer:write"

# Remove a Health Monitor
# DELETE  /v2/lbaas/healthmonitors/{healthmonitor_id}
#"os_load-balancer_api:healthmonitor:delete": "rule:load-balancer:write"

# List L7 Policys
# GET  /v2/lbaas/l7policies
#"os_load-balancer_api:l7policy:get_all": "rule:load-balancer:read"

# List L7 Policys including resources owned by others
# GET  /v2/lbaas/l7policies
#"os_load-balancer_api:l7policy:get_all-global": "rule:load-balancer:read-
↪global"

# Create a L7 Policy
# POST  /v2/lbaas/l7policies
```

(continues on next page)

(continued from previous page)

```
#"os_load-balancer_api:l7policy:post": "rule:load-balancer:write"

# Show L7 Policy details
# GET  /v2/lbaas/l7policies/{l7policy_id}
#"os_load-balancer_api:l7policy:get_one": "rule:load-balancer:read"

# Update a L7 Policy
# PUT  /v2/lbaas/l7policies/{l7policy_id}
#"os_load-balancer_api:l7policy:put": "rule:load-balancer:write"

# Remove a L7 Policy
# DELETE  /v2/lbaas/l7policies/{l7policy_id}
#"os_load-balancer_api:l7policy:delete": "rule:load-balancer:write"

# List L7 Rules
# GET  /v2/lbaas/l7policies/{l7policy_id}/rules
#"os_load-balancer_api:l7rule:get_all": "rule:load-balancer:read"

# Create a L7 Rule
# POST  /v2/lbaas/l7policies/{l7policy_id}/rules
#"os_load-balancer_api:l7rule:post": "rule:load-balancer:write"

# Show L7 Rule details
# GET  /v2/lbaas/l7policies/{l7policy_id}/rules/{l7rule_id}
#"os_load-balancer_api:l7rule:get_one": "rule:load-balancer:read"

# Update a L7 Rule
# PUT  /v2/lbaas/l7policies/{l7policy_id}/rules/{l7rule_id}
#"os_load-balancer_api:l7rule:put": "rule:load-balancer:write"

# Remove a L7 Rule
# DELETE  /v2/lbaas/l7policies/{l7policy_id}/rules/{l7rule_id}
#"os_load-balancer_api:l7rule:delete": "rule:load-balancer:write"

# List Listeners
# GET  /v2/lbaas/listeners
#"os_load-balancer_api:listener:get_all": "rule:load-balancer:read"

# List Listeners including resources owned by others
# GET  /v2/lbaas/listeners
#"os_load-balancer_api:listener:get_all-global": "rule:load-balancer:read-
↪global"

# Create a Listener
# POST  /v2/lbaas/listeners
#"os_load-balancer_api:listener:post": "rule:load-balancer:write"

# Show Listener details
# GET  /v2/lbaas/listeners/{listener_id}
```

(continues on next page)

(continued from previous page)

```
#"os_load-balancer_api:listener:get_one": "rule:load-balancer:read"

# Update a Listener
# PUT  /v2/lbaas/listeners/{listener_id}
#"os_load-balancer_api:listener:put": "rule:load-balancer:write"

# Remove a Listener
# DELETE  /v2/lbaas/listeners/{listener_id}
#"os_load-balancer_api:listener:delete": "rule:load-balancer:write"

# Show Listener statistics
# GET  /v2/lbaas/listeners/{listener_id}/stats
#"os_load-balancer_api:listener:get_stats": "rule:load-balancer:read"

# List Load Balancers
# GET  /v2/lbaas/loadbalancers
#"os_load-balancer_api:loadbalancer:get_all": "rule:load-balancer:read"

# List Load Balancers including resources owned by others
# GET  /v2/lbaas/loadbalancers
#"os_load-balancer_api:loadbalancer:get_all-global": "rule:load-balancer:read-
↪global"

# Create a Load Balancer
# POST  /v2/lbaas/loadbalancers
#"os_load-balancer_api:loadbalancer:post": "rule:load-balancer:write"

# Show Load Balancer details
# GET  /v2/lbaas/loadbalancers/{loadbalancer_id}
#"os_load-balancer_api:loadbalancer:get_one": "rule:load-balancer:read"

# Update a Load Balancer
# PUT  /v2/lbaas/loadbalancers/{loadbalancer_id}
#"os_load-balancer_api:loadbalancer:put": "rule:load-balancer:write"

# Remove a Load Balancer
# DELETE  /v2/lbaas/loadbalancers/{loadbalancer_id}
#"os_load-balancer_api:loadbalancer:delete": "rule:load-balancer:write"

# Show Load Balancer statistics
# GET  /v2/lbaas/loadbalancers/{loadbalancer_id}/stats
#"os_load-balancer_api:loadbalancer:get_stats": "rule:load-balancer:read"

# Show Load Balancer status
# GET  /v2/lbaas/loadbalancers/{loadbalancer_id}/status
#"os_load-balancer_api:loadbalancer:get_status": "rule:load-balancer:read"

# Failover a Load Balancer
# PUT  /v2/lbaas/loadbalancers/{loadbalancer_id}/failover
```

(continues on next page)

(continued from previous page)

```
#"os_load-balancer_api:loadbalancer:put_failover": "rule:load-balancer:admin"

# List Members of a Pool
# GET  /v2/lbaas/pools/{pool_id}/members
#"os_load-balancer_api:member:get_all": "rule:load-balancer:read"

# Create a Member
# POST  /v2/lbaas/pools/{pool_id}/members
#"os_load-balancer_api:member:post": "rule:load-balancer:write"

# Show Member details
# GET  /v2/lbaas/pools/{pool_id}/members/{member_id}
#"os_load-balancer_api:member:get_one": "rule:load-balancer:read"

# Update a Member
# PUT  /v2/lbaas/pools/{pool_id}/members/{member_id}
#"os_load-balancer_api:member:put": "rule:load-balancer:write"

# Remove a Member
# DELETE  /v2/lbaas/pools/{pool_id}/members/{member_id}
#"os_load-balancer_api:member:delete": "rule:load-balancer:write"

# List Pools
# GET  /v2/lbaas/pools
#"os_load-balancer_api:pool:get_all": "rule:load-balancer:read"

# List Pools including resources owned by others
# GET  /v2/lbaas/pools
#"os_load-balancer_api:pool:get_all-global": "rule:load-balancer:read-global"

# Create a Pool
# POST  /v2/lbaas/pools
#"os_load-balancer_api:pool:post": "rule:load-balancer:write"

# Show Pool details
# GET  /v2/lbaas/pools/{pool_id}
#"os_load-balancer_api:pool:get_one": "rule:load-balancer:read"

# Update a Pool
# PUT  /v2/lbaas/pools/{pool_id}
#"os_load-balancer_api:pool:put": "rule:load-balancer:write"

# Remove a Pool
# DELETE  /v2/lbaas/pools/{pool_id}
#"os_load-balancer_api:pool:delete": "rule:load-balancer:write"

# List enabled providers
# GET  /v2/lbaas/providers
#"os_load-balancer_api:provider:get_all": "rule:load-balancer:read"
```

(continues on next page)

(continued from previous page)

```
# List Quotas
# GET  /v2/lbaas/quotas
#"os_load-balancer_api:quota:get_all": "rule:load-balancer:read-quota"

# List Quotas including resources owned by others
# GET  /v2/lbaas/quotas
#"os_load-balancer_api:quota:get_all-global": "rule:load-balancer:read-quota-
↪global"

# Show Quota details
# GET  /v2/lbaas/quotas/{project_id}
#"os_load-balancer_api:quota:get_one": "rule:load-balancer:read-quota"

# Update a Quota
# PUT  /v2/lbaas/quotas/{project_id}
#"os_load-balancer_api:quota:put": "rule:load-balancer:write-quota"

# Reset a Quota
# DELETE  /v2/lbaas/quotas/{project_id}
#"os_load-balancer_api:quota:delete": "rule:load-balancer:write-quota"

# Show Default Quota for a Project
# GET  /v2/lbaas/quotas/{project_id}/default
#"os_load-balancer_api:quota:get_defaults": "rule:load-balancer:read-quota"

# List Amphorae
# GET  /v2/octavia/amphorae
#"os_load-balancer_api:amphora:get_all": "rule:load-balancer:admin"

# Show Amphora details
# GET  /v2/octavia/amphorae/{amphora_id}
#"os_load-balancer_api:amphora:get_one": "rule:load-balancer:admin"

# Delete an Amphora
# DELETE  /v2/octavia/amphorae/{amphora_id}
#"os_load-balancer_api:amphora:delete": "rule:load-balancer:admin"

# Update Amphora Agent Configuration
# PUT  /v2/octavia/amphorae/{amphora_id}/config
#"os_load-balancer_api:amphora:put_config": "rule:load-balancer:admin"

# Failover Amphora
# PUT  /v2/octavia/amphorae/{amphora_id}/failover
#"os_load-balancer_api:amphora:put_failover": "rule:load-balancer:admin"

# Show Amphora statistics
# GET  /v2/octavia/amphorae/{amphora_id}/stats
#"os_load-balancer_api:amphora:get_stats": "rule:load-balancer:admin"
```

(continues on next page)

(continued from previous page)

```
# List the provider flavor capabilities.
# GET  /v2/lbaas/providers/{provider}/flavor_capabilities
#"os_load-balancer_api:provider-flavor:get_all": "rule:load-balancer:admin"

# List the provider availability zone capabilities.
# GET  /v2/lbaas/providers/{provider}/availability_zone_capabilities
#"os_load-balancer_api:provider-availability-zone:get_all": "rule:load-
↪balancer:admin"
```

1.3 Optional Installation and Configuration Guides

1.3.1 Available Provider Drivers

Octavia supports enabling multiple provider drivers via the Octavia v2 API. Drivers, other than the reference Amphora driver, exist outside of the Octavia repository and are not maintained by the Octavia team. This list is intended to provide a place for operators to discover and find available load balancing provider drivers.

This list is a **"best effort"** to keep updated, so please check with your favorite load balancer provider to see if they support OpenStack load balancing. If they don't, make a request for support!

Note: The provider drivers listed here may not be maintained by the OpenStack LBaaS (Octavia) team. Please submit bugs for these projects through their respective bug tracking systems.

Drivers are installed on all of your Octavia API instances using pip and automatically integrated with Octavia using setuptools entry points. Once installed, operators can enable the provider by adding the provider to the Octavia configuration file enabled_provider_drivers setting in the [api_settings] section. Be sure to install and enable the provider on all of your Octavia API instances.

A10 Networks OpenStack Octavia Driver

A10 Networks Octavia Driver for Thunder, vThunder and AX Series Appliances.

Default provider name: **a10**

The driver source: https://github.com/a10networks/a10-octavia/

The documentation: https://github.com/a10networks/a10-octavia/

Where to report issues with the driver: Contact A10 Networks

Amphora

This is the reference driver for Octavia, meaning it is used for testing the Octavia code base. It is an open source, scalable, and highly available load balancing provider.

Default provider name: **amphora**

The driver package: https://pypi.org/project/octavia/

The driver source: https://opendev.org/openstack/octavia/

The documentation: https://docs.openstack.org/octavia/latest/

Where to report issues with the driver: https://storyboard.openstack.org/#!/project/openstack/octavia

Amphorav2

This is extension of the reference driver for Octavia. It adopts taskflow jobboard feature and saves task states into the persistence backend, this allows to continue task execution if controller work was interrupted.

Default provider name: **amphorav2**

The driver package: https://pypi.org/project/octavia/

The driver source: https://opendev.org/openstack/octavia/

The documentation: https://docs.openstack.org/octavia/latest/

Where to report issues with the driver: https://storyboard.openstack.org/#!/project/openstack/octavia

F5 Networks Provider Driver for OpenStack Octavia by SAP SE

F5 Networks Provider Driver for OpenStack Octavia provided by SAP SE.

Default provider name: **f5**

The driver source: https://github.com/sapcc/octavia-f5-provider-driver

Where to report issues with the driver: Contact SAP SE

OVN Octavia Provider Driver

OVN provides virtual networking for Open vSwitch and is a component of the Open vSwitch project. This project provides integration between OpenStack Octavia and OVN.

Default provider name: **ovn**

The driver package: https://pypi.org/project/ovn-octavia-provider/

The driver source: https://opendev.org/openstack/ovn-octavia-provider

The documentation: https://docs.openstack.org/ovn-octavia-provider/latest/

Where to report issues with the driver: https://bugs.launchpad.net/neutron/+bugs?field.tag=ovn-octavia-provider

Radware Provider Driver for OpenStack Octavia

Radware provider driver for OpenStack Octavia.

Default provider name: **radware**

The driver package: https://pypi.org/project/radware_octavia_rocky_driver/

The documentation: https://pypi.org/project/radware_octavia_rocky_driver/

Where to report issues with the driver: Contact Radware

VMware NSX

VMware NSX Octavia Driver.

Default provider name: **vmwareedge**

The driver package: https://pypi.org/project/vmware-nsx/

The driver source: https://opendev.org/x/vmware-nsx

Where to report issues with the driver: https://bugs.launchpad.net/vmware-nsx

1.3.2 Octavia Amphora Log Offloading

The default logging configuration will store the logs locally, on the amphora filesystem with file rotation.

Octavia Amphorae can offload their log files via the syslog protocol to syslog receivers via the load balancer management network (lb-mgmt-net). This allows log aggregation of both administrative logs and also tenant traffic flow logs. The syslog receivers can either be local to the load balancer management network or routable via the load balancer management network. By default any syslog receiver that supports UDP or TCP syslog protocol can be used, however the operator also has the option to create an override rsyslog configuration template to enable other features or protocols their Amphora image may support.

This guide will discuss the features of *Amphora* log offloading and how to configure them.

Administrative Logs

The administrative log offloading feature of the *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 separately.

All administrative log messages will be sent using the native log format for the application sending the message.

Enabling Administrative Log Offloading

One or more syslog receiver endpoints must be configured in the Octavia configuration file to enable administrative log offloading. The first endpoint will be the primary endpoint to receive the syslog packets. Should the first endpoint become unavailable, the additional endpoints listed will be tried one at a time.

Note: Secondary syslog endpoints will only be used if the log_protocol is configured for TCP. With the UDP syslog protocol, rsyslog is unable to detect if the primary endpoint has failed.

To configure administrative log offloading, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[amphora_agent]
admin_log_targets = 192.0.2.1:10514, 2001:db8:1::10:10514
```

In this example, the primary syslog receiver will be 192.0.2.1 on port 10514. The backup syslog receiver will be 2001:db8:1::10 on port 10514.

Note: Make sure your syslog receiver endpoints are accessible from the load balancer management network and you have configured the required security group or firewall rules to allow the traffic. These endpoints can be routable addresses from the load balancer management network.

The load balancer related administrative logs will be sent using a LOG_LOCAL[0-7] facility. The facility number defaults to 1, but is configurable using the administrative_log_facility setting in the Octavia configuration file.

To configure administrative log facility, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[amphora_agent]
administrative_log_facility = 1
```

Forwarding All Administrative Logs

By default, the Amphorae will only forward load balancer related administrative logs, such as the haproxy admin logs, keepalived, and *Amphora* agent logs. You can optionally configure the Amphorae to send all of the administrative logs from the *Amphora*, such as the kernel, system, and security logs. Even with this setting the tenant flow logs will not be included. You can configure tenant flow log forwarding in the *Tenant Flow Logs* section.

The load balancer related administrative logs will be sent using the LOG_LOCAL[0-7] configured using the administrative_log_facility setting. All other administrative log messages will use their native syslog facilities.

To configure the Amphorae to forward all administrative logs, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[amphora_agent]
forward_all_logs = True
```

Tenant Flow Logs

Enabling Tenant Flow Log Offloading

One or more syslog receiver endpoints must be configured in the Octavia configuration file to enable tenant flow log offloading. The first endpoint will be the primary endpoint to receive the syslog packets. Should the first endpoint become unavailable, the additional endpoints listed will be tried one at a time. The endpoints configured for tenant flow log offloading may be the same endpoints as the administrative log offloading configuration.

> **Warning:** Tenant flow logging can produce a large number of syslog messages depending on how many connections the load balancers are receiving. Tenant flow logging produces one log entry per connection to the load balancer. We recommend you monitor, size, and configure your syslog receivers appropriately based on the expected number of connections your load balancers will be handling.

Note: Secondary syslog endpoints will only be used if the log_protocol is configured for TCP. With the UDP syslog protocol, rsyslog is unable to detect if the primary endpoint has failed.

To configure tenant flow log offloading, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[amphora_agent]
tenant_log_targets = 192.0.2.1:10514, 2001:db8:1::10:10514
```

In this example, the primary syslog receiver will be 192.0.2.1 on port 10514. The backup syslog receiver will be 2001:db8:1::10 on port 10514.

Note: Make sure your syslog receiver endpoints are accessible from the load balancer management network and you have configured the required security group or firewall rules to allow the traffic. These endpoints can be routable addresses from the load balancer management network.

The load balancer related tenant flow logs will be sent using a LOG_LOCAL[0-7] facility. The facility number defaults to 0, but is configurable using the user_log_facility setting in the Octavia configuration file.

To configure the tenant flow log facility, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[amphora_agent]
user_log_facility = 0
```

Tenant Flow Log Format

The default tenant flow log format is:

```
project_id loadbalancer_id listener_id client_ip client_port data_time
request_string http_status bytes_read bytes_uploaded
client_certificate_verify(0 or 1) client_certificate_distinguised_name
pool_id member_id processing_time(ms) termination_state
```

Any field that is unknown or not applicable to the connection will have a '-' character in its place.

An example log entry when using rsyslog as the syslog receiver is:

Note: The prefix[1] in this example comes from the rsyslog receiver and is not part of the syslog message from the amphora.

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

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

Custom Tenant Flow Log Format

You can optionally specify a custom log format for the tenant flow logs. This string follows the HAProxy log format variables with the exception of the "{{ project_id }}" and "{{ lb_id }}" variables that will be replaced by the Octavia *Amphora* driver. These custom variables are optional.

See the HAProxy documentation for Custom log format variable definitions.

To configure a custom log format, set the following setting in your Octavia configuration file for all of the controllers and restart them:

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

Disabling Logging

There may be cases where you need to disable logging inside the *Amphora*, such as complying with regulatory standards. Octavia provides multiple options for disabling *Amphora* logging.

Disable Local Log Storage

This setting stops log entries from being written to the disk inside the *Amphora*. Logs can still be sent via *Amphora* log offloading if log offloading is configured for the Amphorae. Enabling this setting may provide a performance benefit to the load balancer.

Warning: This feature disables ALL log storage in the *Amphora*, including kernel, system, and security logging.

Note: If you enable this setting and are not using *Amphora* log offloading, we recommend you also *Disable Tenant Flow Logging* to improve load balancing performance.

To disable local log storage in the *Amphora*, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[amphora_agent]
disable_local_log_storage = True
```

Disable Tenant Flow Logging

This setting allows you to disable tenant flow logging irrespective of the other logging configuration settings. It will take precedent over the other settings. When this setting is enabled, no tenant flow (connection) logs will be written to the disk inside the *Amphora* or be sent via the *Amphora* log offloading.

Note: Disabling tenant flow logging can also improve the load balancing performance of the amphora. Due to the potential performance improvement, we recommend you enable this setting when using the *Disable Local Log Storage* setting.

To disable tenant flow logging, set the following setting in your Octavia configuration file for all of the controllers and restart them:

```
[haproxy_amphora]
connection_logging = False
```

1.3.3 Octavia API Auditing

The keystonemiddleware audit middleware supports delivery of Cloud Auditing Data Federation (CADF) audit events via Oslo messaging notifier capability. Based on *notification_driver* configuration, audit events can be routed to messaging infrastructure (notification_driver = messagingv2) or can be routed to a log file (notification_driver = log).

More information about the CADF format can be found on the DMTF Cloud Auditing Data Federation website.

Audit middleware creates two events per REST API interaction. First event has information extracted from request data and the second one has request outcome (response).

Configuring Octavia API Auditing

Auditing can be enabled by making the following changes to the Octavia configuration file on your Octavia API instance(s).

1. Enable auditing:

   ```
   [audit]
   ...
   enabled = True
   ```

2. Optionally specify the location of the audit map file:

   ```
   [audit]
   ...
   audit_map_file = /etc/octavia/octavia_api_audit_map.conf
   ```

   The default audit map file location is /etc/octavia/octavia_api_audit_map.conf.

3. Copy the audit map file from the octavia/etc/audit directory to the location specified in the previous step. A sample file has been provided in octavia/etc/audit/octavia_api_audit_map.conf.sample.

4. Optionally specify the REST HTTP methods you do not want to audit:

   ```
   [audit]
   ...
   ignore_req_list =
   ```

5. Specify the driver to use for sending the audit notifications:

   ```
   [audit_middleware_notifications]
   ...
   driver = log
   ```

   Driver options are: messaging, messagingv2, routing, log, noop

6. Optionally specify the messaging topic:

   ```
   [audit_middleware_notifications]
   ...
   topics =
   ```

7. Optionally specify the messaging transport URL:

   ```
   [audit_middleware_notifications]
   ...
   transport_url =
   ```

8. Restart your Octavia API processes.

Sampe Audit Events

Request

```
{
  "event_type": "audit.http.request",
  "timestamp": "2018-10-11 22:42:22.721025",
  "payload": {
    "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
    "eventTime": "2018-10-11T22:42:22.720112+0000",
    "target": {
      "id": "octavia",
      "typeURI": "service/load-balancer/loadbalancers",
      "addresses": [{
        "url": "http://10.21.21.53/load-balancer",
        "name": "admin"
      }, {
        "url": "http://10.21.21.53/load-balancer",
        "name": "private"
      }, {
        "url": "http://10.21.21.53/load-balancer",
        "name": "public"
      }],
      "name": "octavia"
    },
    "observer": {
      "id": "target"
    },
    "tags": ["correlation_id?value=e5b34bc3-4837-54fa-9892-8e65a9a2e73a"],
    "eventType": "activity",
    "initiator": {
      "typeURI": "service/security/account/user",
      "name": "admin",
      "credential": {
        "token": "***",
        "identity_status": "Confirmed"
      },
      "host": {
        "agent": "openstacksdk/0.17.2 keystoneauth1/3.11.0 python-requests/2.19.1 CPython/2.7.12",
        "address": "10.21.21.53"
      },
      "project_id": "90168d185e504b5580884a235ba31612",
      "id": "2af901396a424d5ca9dffa725226e8c7"
    },
    "action": "read/list",
    "outcome": "pending",
    "id": "8cf14af5-246e-5739-a11e-513ca13b7d36",
    "requestPath": "/load-balancer/v2.0/lbaas/loadbalancers"
  },
```

(continues on next page)

(continued from previous page)

```
  "priority": "INFO",
  "publisher_id": "uwsgi",
  "message_id": "63264e0e-e60f-4adc-a656-0d87ab5d6329"
}
```

Response

```
{
  "event_type": "audit.http.response",
  "timestamp": "2018-10-11 22:42:22.853129",
  "payload": {
    "typeURI": "http://schemas.dmtf.org/cloud/audit/1.0/event",
    "eventTime": "2018-10-11T22:42:22.720112+0000",
    "target": {
      "id": "octavia",
      "typeURI": "service/load-balancer/loadbalancers",
      "addresses": [{
        "url": "http://10.21.21.53/load-balancer",
        "name": "admin"
      }, {
        "url": "http://10.21.21.53/load-balancer",
        "name": "private"
      }, {
        "url": "http://10.21.21.53/load-balancer",
        "name": "public"
      }],
      "name": "octavia"
    },
    "observer": {
      "id": "target"
    },
    "tags": ["correlation_id?value=e5b34bc3-4837-54fa-9892-8e65a9a2e73a"],
    "eventType": "activity",
    "initiator": {
      "typeURI": "service/security/account/user",
      "name": "admin",
      "credential": {
        "token": "***",
        "identity_status": "Confirmed"
      },
      "host": {
        "agent": "openstacksdk/0.17.2 keystoneauth1/3.11.0 python-requests/2.
↪19.1 CPython/2.7.12",
        "address": "10.21.21.53"
      },
      "project_id": "90168d185e504b5580884a235ba31612",
      "id": "2af901396a424d5ca9dffa725226e8c7"
    },
```

(continues on next page)

(continued from previous page)

```
    "reason": {
      "reasonCode": "200",
      "reasonType": "HTTP"
    },
    "reporterchain": [{
      "reporterTime": "2018-10-11T22:42:22.852613+0000",
      "role": "modifier",
      "reporter": {
        "id": "target"
      }
    }],
    "action": "read/list",
    "outcome": "success",
    "id": "8cf14af5-246e-5739-a11e-513ca13b7d36",
    "requestPath": "/load-balancer/v2.0/lbaas/loadbalancers"
  },
  "priority": "INFO",
  "publisher_id": "uwsgi",
  "message_id": "7cd89dce-af6e-40c5-8634-e87d1ed32a3c"
}
```

1.3.4 Octavia API Health Monitoring

The Octavia API provides a health monitoring endpoint that can be used by external load balancers to manage the Octavia API pool. When properly configured, the health monitoring endpoint will reflect the full operational status of the Octavia API.

The Octavia API health monitoring endpoint extends the OpenStack Oslo middleware healthcheck library to test the Octavia Pecan API framework and associated services.

Oslo Healthcheck Queries

Oslo middleware healthcheck supports HTTP **"GET"** and **"HEAD"** methods.

The response from Oslo middleware healthcheck can be customized by specifying the acceptable response type for the request.

Oslo middleware healthcheck currently supports the following types:

- text/plain
- text/html
- application/json

If the requested type is not one of the above, it defaults to text/plain.

Note: The content of the response "reasons" will vary based on the backend plugins enabled in Oslo middleware healthcheck. It is a best practice to only rely on the HTTP status code for Octavia API health monitoring.

Example Responses

Example passing output for text/plain with *detailed* False:

```
$ curl -i http://198.51.100.10/load-balancer/healthcheck

  HTTP/1.1 200 OK
  Date: Mon, 16 Mar 2020 18:10:27 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/plain; charset=UTF-8
  Content-Length: 2
  x-openstack-request-id: req-9c6f4303-63a7-4f30-8afc-39340658702f
  Connection: close
  Vary: Accept-Encoding

  OK
```

Example failing output for text/plain with *detailed* False:

```
$ curl -i http://198.51.100.10/load-balancer/healthcheck

  HTTP/1.1 503 Service Unavailable
  Date: Mon, 16 Mar 2020 18:42:12 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/plain; charset=UTF-8
  Content-Length: 36
  x-openstack-request-id: req-84024269-2dfb-41ad-bfda-b3e1da138bba
  Connection: close
```

Example passing output for text/html with *detailed* False:

```
$ curl -i -H "Accept: text/html" http://198.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 200 OK
  Date: Mon, 16 Mar 2020 18:25:11 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/html; charset=UTF-8
  Content-Length: 239
  x-openstack-request-id: req-b212d619-146f-4b50-91a3-5da16051badc
  Connection: close
  Vary: Accept-Encoding

  <HTML>
  <HEAD><TITLE>Healthcheck Status</TITLE></HEAD>
  <BODY>

  <H2>Result of 1 checks:</H2>
  <TABLE bgcolor="#ffffff" border="1">
  <TBODY>
  <TR>
```

(continues on next page)

(continued from previous page)

```

  <TH>
  Reason
  </TH>
  </TR>
  <TR>

      <TD>OK</TD>

  </TR>
  </TBODY>
  </TABLE>
  <HR></HR>

  </BODY>
  </HTML>
```

Example failing output for text/html with *detailed* False:

```
$ curl -i -H "Accept: text/html" http://198.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 503 Service Unavailable
  Date: Mon, 16 Mar 2020 18:42:22 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/html; charset=UTF-8
  Content-Length: 273
  x-openstack-request-id: req-c91dd214-85ca-4d33-9fa3-2db81566d9e5
  Connection: close

  <HTML>
  <HEAD><TITLE>Healthcheck Status</TITLE></HEAD>
  <BODY>

  <H2>Result of 1 checks:</H2>
  <TABLE bgcolor="#ffffff" border="1">
  <TBODY>
  <TR>

  <TH>
  Reason
  </TH>
  </TR>
  <TR>

      <TD>The Octavia database is unavailable.</TD>

  </TR>
  </TBODY>
```

(continues on next page)

(continued from previous page)

```
  </TABLE>
  <HR></HR>

  </BODY>
  </HTML>
```

Example passing output for application/json with *detailed* False:

```
$ curl -i -H "Accept: application/json" http://192.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 200 OK
  Date: Mon, 16 Mar 2020 18:34:42 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: application/json
  Content-Length: 62
  x-openstack-request-id: req-417dc85c-e64e-496e-a461-494a3e6a5479
  Connection: close

  {
      "detailed": false,
      "reasons": [
          "OK"
      ]
  }
```

Example failing output for application/json with *detailed* False:

```
$ curl -i -H "Accept: application/json" http://192.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 503 Service Unavailable
  Date: Mon, 16 Mar 2020 18:46:28 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: application/json
  Content-Length: 96
  x-openstack-request-id: req-de50b057-6105-4fca-a758-c872ef28bbfa
  Connection: close

  {
      "detailed": false,
      "reasons": [
          "The Octavia database is unavailable."
      ]
  }
```

Example Detailed Responses

Example passing output for text/plain with *detailed* True:

```
$ curl -i http://198.51.100.10/load-balancer/healthcheck

  HTTP/1.1 200 OK
  Date: Mon, 16 Mar 2020 18:10:27 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/plain; charset=UTF-8
  Content-Length: 2
  x-openstack-request-id: req-9c6f4303-63a7-4f30-8afc-39340658702f
  Connection: close
  Vary: Accept-Encoding

  OK
```

Example failing output for text/plain with *detailed* True:

```
$ curl -i http://198.51.100.10/load-balancer/healthcheck

  HTTP/1.1 503 Service Unavailable
  Date: Mon, 16 Mar 2020 23:41:23 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/plain; charset=UTF-8
  Content-Length: 36
  x-openstack-request-id: req-2cd046cb-3a6c-45e3-921d-5f4a9e65c63e
  Connection: close
```

Example passing output for text/html with *detailed* True:

```
$ curl -i -H "Accept: text/html" http://198.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 200 OK
  Date: Mon, 16 Mar 2020 22:11:54 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/html; charset=UTF-8
  Content-Length: 9927
  x-openstack-request-id: req-ae7404c9-b183-46dc-bb1b-e5f4e4984a57
  Connection: close
  Vary: Accept-Encoding

  <HTML>
  <HEAD><TITLE>Healthcheck Status</TITLE></HEAD>
  <BODY>
  <H1>Server status</H1>
  <B>Server hostname:</B><PRE>devstack2</PRE>
  <B>Current time:</B><PRE>2020-03-16 22:11:54.320529</PRE>
  <B>Python version:</B><PRE>3.6.9 (default, Nov  7 2019, 10:44:02)
  [GCC 8.3.0]</PRE>
```

(continues on next page)

(continued from previous page)

```
<B>Platform:</B><PRE>Linux-4.15.0-88-generic-x86_64-with-Ubuntu-18.04-bionic
↪</PRE>
<HR></HR>
<H2>Garbage collector:</H2>
<B>Counts:</B><PRE>(28, 10, 4)</PRE>
<B>Thresholds:</B><PRE>(700, 10, 10)</PRE>
<HR></HR>
<H2>Result of 1 checks:</H2>
<TABLE bgcolor="#ffffff" border="1">
<TBODY>
<TR>
<TH>
Kind
</TH>
<TH>
Reason
</TH>
<TH>
Details
</TH>

</TR>
<TR>
<TD>OctaviaDBCheckResult</TD>
    <TD>OK</TD>
<TD></TD>
</TR>
</TBODY>
</TABLE>
<HR></HR>
<H2>1 greenthread(s) active:</H2>
<TABLE bgcolor="#ffffff" border="1">
<TBODY>
<TR>
    <TD><PRE> <...> </PRE></TD>
</TR>
</TBODY>
</TABLE>
<HR></HR>
<H2>1 thread(s) active:</H2>
<TABLE bgcolor="#ffffff" border="1">
<TBODY>
<TR>
    <TD><PRE> <...> </PRE></TD>
</TR>
</TBODY>
</TABLE>
</BODY>
</HTML>
```

Example failing output for text/html with *detailed* True:

```
$ curl -i -H "Accept: text/html" http://198.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 503 Service Unavailable
  Date: Mon, 16 Mar 2020 23:43:52 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: text/html; charset=UTF-8
  Content-Length: 10211
  x-openstack-request-id: req-39b65058-6dc3-4069-a2d5-8a9714dba61d
  Connection: close

  <HTML>
  <HEAD><TITLE>Healthcheck Status</TITLE></HEAD>
  <BODY>
  <H1>Server status</H1>
  <B>Server hostname:</B><PRE>devstack2</PRE>
  <B>Current time:</B><PRE>2020-03-16 23:43:52.411127</PRE>
  <B>Python version:</B><PRE>3.6.9 (default, Nov  7 2019, 10:44:02)
  [GCC 8.3.0]</PRE>
  <B>Platform:</B><PRE>Linux-4.15.0-88-generic-x86_64-with-Ubuntu-18.04-bionic
↪</PRE>
  <HR></HR>
  <H2>Garbage collector:</H2>
  <B>Counts:</B><PRE>(578, 10, 4)</PRE>
  <B>Thresholds:</B><PRE>(700, 10, 10)</PRE>
  <HR></HR>
  <H2>Result of 1 checks:</H2>
  <TABLE bgcolor="#ffffff" border="1">
  <TBODY>
  <TR>
  <TH>
  Kind
  </TH>
  <TH>
  Reason
  </TH>
  <TH>
  Details
  </TH>

  </TR>
  <TR>
  <TD>OctaviaDBCheckResult</TD>
      <TD>The Octavia database is unavailable.</TD>
  <TD>Database health check failed due to: (pymysql.err.OperationalError)␣
↪(2003, &#34;Can&#39;t connect to MySQL server on &#39;127.0.0.1&#39;␣
↪([Errno 111] Connection refused)&#34;)
  [SQL: SELECT 1]
  (Background on this error at: http://sqlalche.me/e/e3q8).</TD>
```

(continues on next page)

(continued from previous page)

```
  </TR>
  </TBODY>
  </TABLE>
  <HR></HR>
  <H2>1 greenthread(s) active:</H2>
  <TABLE bgcolor="#ffffff" border="1">
  <TBODY>
  <TR>
      <TD><PRE> <...> </PRE></TD>
  </TR>
  </TBODY>
  </TABLE>
  <HR></HR>
  <H2>1 thread(s) active:</H2>
  <TABLE bgcolor="#ffffff" border="1">
  <TBODY>
  <TR>
      <TD><PRE> <...> </PRE></TD>
  </TR>
  </TBODY>
  </TABLE>
  </BODY>
  </HTML>
```

Example passing output for application/json with *detailed* True:

```
$ curl -i -H "Accept: application/json" http://192.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 200 OK
  Date: Mon, 16 Mar 2020 22:05:26 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: application/json
  Content-Length: 9298
  x-openstack-request-id: req-d3913655-6e3f-4086-a252-8bb297ea5fd6
  Connection: close

  {
      "detailed": true,
      "gc": {
          "counts": [
              27,
              10,
              4
          ],
          "threshold": [
              700,
              10,
              10
```

(continues on next page)

(continued from previous page)

```
          ]
      },
      "greenthreads": [
          <...>
      ],
      "now": "2020-03-16 22:05:26.431429",
      "platform": "Linux-4.15.0-88-generic-x86_64-with-Ubuntu-18.04-bionic",
      "python_version": "3.6.9 (default, Nov  7 2019, 10:44:02) \n[GCC 8.3.0]
↪",
      "reasons": [
          {
              "class": "OctaviaDBCheckResult",
              "details": "",
              "reason": "OK"
          }
      ],
      "threads": [
          <...>
      ]
  }
```

Example failing output for application/json with *detailed* True:

```
$ curl -i -H "Accept: application/json" http://192.51.100.10/load-balancer/
↪healthcheck

  HTTP/1.1 503 Service Unavailable
  Date: Mon, 16 Mar 2020 23:56:43 GMT
  Server: Apache/2.4.29 (Ubuntu)
  Content-Type: application/json
  Content-Length: 9510
  x-openstack-request-id: req-3d62ea04-9bdb-4e19-b218-1a81ff7d7337
  Connection: close

  {
      "detailed": true,
      "gc": {
          "counts": [
              178,
              0,
              5
          ],
          "threshold": [
              700,
              10,
              10
          ]
      },
      "greenthreads": [
```

(continues on next page)

(continued from previous page)

```
            <...>
        ],
        "now": "2020-03-16 23:58:23.361209",
        "platform": "Linux-4.15.0-88-generic-x86_64-with-Ubuntu-18.04-bionic",
        "python_version": "3.6.9 (default, Nov  7 2019, 10:44:02) \n[GCC 8.3.0]
↪",
        "reasons": [
            {
                "class": "OctaviaDBCheckResult",
                "details": "(pymysql.err.OperationalError) (2003, \"Can't␣
↪connect to MySQL server on '127.0.0.1' ([Errno 111] Connection refused)\")\
↪n(Background on this error at: http://sqlalche.me/e/e3q8)",
                "reason": "The Octavia database is unavailable."
            }
        ],
        "threads": [
            <...>
        ]
    }
```

Oslo Healthcheck Plugins

The Octavia API health monitoring endpoint, implemented with Oslo middleware healthcheck, is extensible using optional backend plugins. There are currently plugins provided by the Oslo middleware library and plugins provided by Octavia.

Oslo middleware provided plugins

- disable_by_file
- disable_by_files_ports

Octavia provided plugins

- *octavia_db_check*

> **Warning:** Some plugins may have long timeouts. It is a best practice to configure your healthcheck query to have connection, read, and/or data timeouts. The appropriate values will be unique to each deployment depending on the cloud performance, number of plugins, etc.

Enabling Octavia API Health Monitoring

To enable the Octavia API health monitoring endpoint, the proper configuration file settings need to be updated and the Octavia API processes need to be restarted.

Start by enabling the endpoint:

```
[api_settings]
healthcheck_enabled = True
```

When the healthcheck_enabled setting is *False*, queries of the /healthcheck will receive an HTTP 404 Not Found response.

You will then need to select the desired monitoring backend plugins:

```
[healthcheck]
backends = octavia_db_check
```

Note: When no plugins are configured, the behavior of Oslo middleware healthcheck changes. Not only does it not run any tests, it will return 204 results instead of 200.

The Octavia API health monitoring endpoint does not require a keystone token for access to allow external load balancers to query the endpoint. For this reason we recommend you restrict access to it on your external load balancer to prevent abuse.

As an additional protection, the API will cache results for a configurable period of time. This means that queries to the health monitoring endpoint will return cached results until the refresh interval has expired, at which point the health check plugin will rerun the check.

By default, the refresh interval is five seconds. This can be configured by adjusting the healthcheck_refresh_interval setting in the Octavia configuration file:

```
[api_settings]
healthcheck_refresh_interval = 5
```

Optionally you can enable the "detailed" mode in Oslo middleware healthcheck. This will cause Oslo middleware healthcheck to return additional information about the API instance. It will also provide exception details if one was raised during the health check. This setting is False and disabled by default in the Octavia API.

```
[healthcheck]
detailed = True
```

Warning: Enabling the 'detailed' setting will expose sensitive details about the API process. Do not enabled this unless you are sure it will not pose a **security risk** to your API instances. We highly recommend you do not enable this.

Using Octavia API Health Monitoring

The Octavia API health monitoring endpoint can be accessed via the /healthmonitor path on the Octavia API endpoint.

For example, if your Octavia (load-balancer) endpoint in keystone is:

```
https://10.21.21.78/load-balancer
```

You would access the Octavia API health monitoring endpoint via:

```
https://10.21.21.78/load-balancer/healthcheck
```

A keystone token is not required to access this endpoint.

Octavia Plugins

octavia_db_check

The octavia_db_check plugin validates the API instance has a working connection to the Octavia database. It executes a SQL no-op query, 'SELECT 1;', against the database.

Note: Many OpenStack services and libraries, such as oslo.db and sqlalchemy, also use the no-op query, 'SELECT 1;' for health checks.

The possible octavia_db_check results are:

Request	Result	Status Code	"reason" Message
GET	Pass	200	OK
HEAD	Pass	204	
GET	Fail	503	The Octavia database is unavailable.
HEAD	Fail	503	

When running Oslo middleware healthcheck in "detailed" mode, the "details" field will have additional information about the error encountered, including the exception details if they were available.

1.3.5 Octavia Flavors

Octavia flavors are a powerful tool for operators to bring enhanced load balancing capabilities to their users. An Octavia flavor is a predefined set of provider configuration options that are created by the operator. When an user requests a load balancer they can request the load balancer be built with one of the defined flavors. Flavors are defined per provider driver and expose the unique capabilities of each provider.

This document is intended to explain the flavors capability for operators that wish to create flavors for their users.

There are three steps to creating a new Octavia flavor:

1. Decide on the provider flavor capabilities that will be configured in the flavor.
2. Create the flavor profile with the flavor capabilities.
3. Create the user facing flavor.

Provider Capabilities

To start the process of defining a flavor, you will want to look at the flavor capabilities that the provider driver exposes. To do this you can use the provider driver flavor capabilities API or the OpenStack client.

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

With the default RBAC policy, this command is only available to administrators.

This will list all of the flavor capabilities the provider supports and may be configured via a flavor.

As an example, the amphora provider supports the *loadbalancer_topology* capability, among many others:

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

Flavor Profiles

The next step in the process of creating a flavor is to define a flavor profile. The flavor profile includes the provider and the flavor data. The flavor capabilities are the supported flavor data settings for a given provider. A flavor profile can be created using the flavor profile API or the OpenStack client.

For example, to create a flavor for the amphora provider, we would create the following flavor profile:

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

With the default RBAC policy, this command is only available to administrators.

This will create a flavor profile for the amphora provider that creates a load balancer with a single amphora. When you create a flavor profile, the settings are validated with the provider to make sure the provider can support the capabilities specified.

The output of the command above is:

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

Flavors

Finally we will create the user facing Octavia flavor. This defines the information users will see and use to create a load balancer with an Octavia flavor. The name of the flavor is the term users can use when creating a load balancer. We encourage you to include a detailed description for users to clearly understand the capabilities of the flavor you are providing.

To continue the example above, to create a flavor with the flavor profile we created in the previous step we call:

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

(continues on next page)

(continued from previous page)

This will create a user visible Octavia flavor that will create a load balancer that uses one amphora and is not highly available. Users can specify this flavor when creating a new load balancer. Disabled flavors are still visible to users, but they will not be able to create a load balancer using the flavor.

The output of the command above is:

```
+-------------------+--------------------------------------+
| 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.                |
+-------------------+--------------------------------------+
```

At this point, the flavor is available for use by users creating new load balancers.

1.3.6 Running Octavia in Apache

To run Octavia in apache2, copy the `httpd/octavia-api.conf` sample configuration file to the appropriate location for the Apache server.

On Debian/Ubuntu systems it is:

```
/etc/apache2/sites-available/octavia-api.conf
```

Restart Apache to have it start serving Octavia.

1.4 Maintanence and Operations

1.4.1 Operator Maintenance Guide

This document is intended for operators. For a developer guide see the *Developer / Operator Quick Start Guide* in this documentation repository. For an end-user guide, please see the *Basic Load Balancing Cookbook* in this documentation repository.

Monitoring

Monitoring Load Balancer Amphora

Octavia will monitor the load balancing amphorae itself and initiate failovers and/or replacements if they malfunction. Therefore, most installations won't need to monitor the amphorae running the load balancer.

Octavia will log each failover to the corresponding health manager logs. It is advisable to use log analytics to monitor failover trends to notice problems in the OpenStack installation early. We have seen neutron (network) connectivity issues, Denial of Service attacks, and nova (compute) malfunctions lead to a higher than normal failover rate. Alternatively, the monitoring of the other services showed problems as well, so depending on your overall monitoring strategy this might be optional.

If additional monitoring is necessary, review the corresponding calls on the amphora agent REST interface (see *Octavia HAProxy Amphora API*)

Monitoring Pool Members

Octavia will use the health information from the underlying load balancing subsystems to determine the health of members. This information will be streamed to the Octavia database and made available via the status tree or other API methods. For critical applications we recommend to poll this information in regular intervals.

Monitoring Load Balancers

You should monitor the provisioning status of a load balancer, and send alerts if the provisioning status is not ACTIVE. Alerts should not be triggered 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 neutron 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.

A load balancer in a PENDING provisioning status is immutable, it cannot be updated or deleted by another process, this PENDING status acts as a lock on the resource. If a database outage occurs while a load balancer is deleted, created or updated, the Octavia control plane will try to remove the PENDING status and set it to ERROR during a long period of time (around 2h45min with the default settings), to prevent the resource from remaining immutable.

Monitoring load balancer functionality

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 Octavia that might impact the functionality of your load balancer, such as router failures, network connectivity issues, and so on.

Monitoring Octavia Control Plane

To monitor the Octavia control plane we recommend process monitoring of the main Octavia processes:

- octavia-api
- octavia-worker
- octavia-health-manager
- octavia-housekeeping

The Monasca project has a plugin for such monitoring (see Monasca Octavia plugin). Please refer to this project for further information.

Octavia's control plane components are shared nothing and can be scaled linearly. For high availability of the control plane we recommend to run at least one set of components in each availability zone. Furthermore, the octavia-api endpoint could be behind a load balancer or other HA technology. That said, if one or more components fail the system will still be available (though potentially degraded). For instance if you have installed one set of components in each of the three availability zones even if you lose a whole zone Octavia will still be responsive and available - only if you lose the Octavia control plane in all three zones will the service be unavailable. Please note this only addresses control plane availability; the availability of the load balancing function depends highly on the chosen topology and the anti-affinity settings. See our forthcoming HA guide for more details.

Additionally, we recommend to monitor the Octavia API endpoint(s). There currently is no special url to use so just polling the root URL in regular intervals is sufficient.

There is a host of information in the log files which can be used for log analytics. A few examples of what could be monitored are:

- Amphora Build Rate - to determine load of the system
- Amphora Build Time - to determine how long it takes to build an amphora
- Failures/Errors - to be notified of system problems early

Rotating the Amphora Images

Octavia will start load balancers with a pre-built image which contain the amphora agent, a load balancing application, and are seeded with cryptographic certificates through the config drive at start up.

Rotating the image means making a load balancer amphora running with an old image failover to an amphora with a new image. This should be without any measurable interruption in the load balancing functionality when using ACTIVE/STANDBY topology. Standalone load balancers might experience a short outage.

Here are some reasons you might need to rotate the amphora image:

- There has been a (security) update to the underlying operating system
- You want to deploy a new version of the amphora agent or haproxy
- The cryptographic certificates and/or keys on the amphora have been compromised.
- Though not related to rotating images, this procedure might be invoked if you are switching to a different flavor for the underlying virtual machine.

Preparing a New Amphora Image

To prepare a new amphora image you will need to use diskimage-create.sh as described in the README in the diskimage-create directory.

For instance, in the `octavia/diskimage-create` directory, run:

```
./diskimage-create.sh
```

Once you have created a new image you will need to upload it into glance. The following shows how to do this if you have set the image tag in the Octavia configuration file. Make sure to use a user with the same tenant as the Octavia service account:

```
openstack image create --file amphora-x64-haproxy.qcow2 \
--disk-format qcow2 --tag <amphora-image-tag> --private \
--container-format bare /var/lib/octavia/amphora-x64-haproxy.qcow2
```

If you didn't configure image tags and instead configured an image id, you will need to update the Octavia configuration file with the new id and restart the Octavia services (except octavia-api).

Generating a List of Load Balancers to Rotate

The easiest way to generate a list, is to just list the IDs of all load balancers:

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

Take note of the IDs.

Rotating a Load Balancer

Octavia has an API call to initiate the failover of a load balancer:

```
openstack loadbalancer failover <loadbalancer id>
```

You can observe the failover by querying octavia `openstack load balancer show <loadbalancer id>` until the load balancer goes `ACTIVE` again.

Best Practices/Optimizations

Since a failover puts significant load on the OpenStack installation by creating new virtual machines and ports, it should either be done at a very slow pace, during a time with little load, or with the right throttling enabled in Octavia. The throttling will make sure to prioritize failovers higher than other operations and depending on how many failovers are initiated this might crowd out other operations.

Rotating Cryptographic Certificates

Octavia secures the communication between the amphora agent and the control plane with two-way SSL encryption. To accomplish that, several certificates are distributed in the system:

- Control plane:
  - Amphora certificate authority (CA) certificate: Used to validate amphora certificates if Octavia acts as a Certificate Authority to issue new amphora certificates
  - Client certificate: Used to authenticate with the amphora
- Amphora:
  - Client CA certificate: Used to validate control plane client certificate
  - Amphora certificate: Presented to control plane processes to prove amphora identity.

The heartbeat UDP packets emitted from the amphora are secured with a symmetric encryption key. This is set by the configuration option *heartbeat_key* in the *health_manager* section. We recommend setting it to a random string of a sufficient length.

Rotating Amphora Certificates

For the server part Octavia will act as a certificate authority itself to issue amphora certificates to be used by each amphora. Octavia will also monitor those certificates and refresh them before they expire.

There are three ways to initiate a rotation manually:

- Change the expiration date of the certificate in the database. Octavia will then rotate the amphora certificates with newly issued ones. This requires the following:
  - Client CA certificate hasn't expired or the corresponding client certificate on the control plane hasn't been issued by a different client CA (in case the authority was compromised)
  - The Amphora CA certificate on the control plane didn't change in any way which jeopardizes validation of the amphora certificate (e.g. the certificate was reissued with a new private/public key)

- If the amphora CA changed in a way which jeopardizes validation of the amphora certificate an operator can manually upload newly issued amphora certificates by switching off validation of the old amphora certificate. This requires a client certificate which can be validated by the client CA file on the amphora. Refer to *Octavia HAProxy Amphora API* for more details.
- If the client certificate on the control plane changed in a way that it can't be validated by the client certificate authority certificate on the amphora, a failover (see *Rotating Amphora Certificates*) of all amphorae needs to be initiated. Until the failover is completed the amphorae can't be controlled by the control plane.

Rotating the Certificate Authority Certificates

If there is a compromise of the certificate authorities' certificates, or they expired, new ones need to be installed into the system. If Octavia is not acting as the certificate authority only the certificate authority's cert needs to be changed in the system so amphora can be authenticated again.

- Issue new certificates (see the script in the bin folder of Octavia if Octavia is acting as the certificate authority) or follow the instructions of the third-party certificate authority. Copy the certificate and the private key (if Octavia acts as a certificate authority) where Octavia can find them.
- If the previous certificate files haven't been overridden, adjust the paths to the new certs in the configuration file and restart all Octavia services (except octavia-api).

Review *Rotating Amphora Certificates* above to determine if and how the amphora certificates needs to be rotated.

Rotating Client Certificates

If the client certificates expired new ones need to be issued and installed on the system:

- Issue a new client certificate (see the script in the bin folder of Octavia if self signed certificates are used) or use the ones provided to you by your certificate authority.
- Copy the new cert where Octavia can find it.
- If the previous certificate files haven't been overridden, adjust the paths to the new certs in the configuration file. In all cases restart all Octavia services except octavia-api.

If the client CA certificate has been replaced in addition to rotating the client certificate the new client CA certificate needs to be installed in the system. After that initiate a failover of all amphorae to distribute the new client CA cert. Until the failover is completed the amphorae can't be controlled by the control plane.

Changing The Heartbeat Encryption Key

Special caution needs to be taken to replace the heartbeat encryption key. Once this is changed Octavia can't read any heartbeats and will assume all amphora are in an error state and initiate an immediate failover.

In preparation, read the chapter on *Best Practices/Optimizations* in the Failover section.

Given the risks involved with changing this key it should not be changed during routine maintenance but only when a compromise is strongly suspected.

Note: For future versions of Octavia an "update amphora" API is planned which will allow this key to be changed without failover. At that time there would be a procedure to halt health monitoring while the keys are rotated and then resume health monitoring.

Handling a VM Node Failure

If a node fails which is running amphora, Octavia will automatically failover the amphora to a different node (capacity permitting). In some cases, the node can be recovered (e.g. through a hard reset) and the hypervisor might bring back the amphora vms. In this case, an operator should manually delete all amphora on this specific node since Octavia assumes they have been deleted as part of the failover and will not touch them again.

Note: As a safety measure an operator can, prior to deleting, manually check if the VM is in use. First, use the Amphora API to obtain the current list of amphorae, then match the nova instance ID to the compute_id column in the amphora API response (it is not currently possible to filter amphora by compute_id). If there are any matches where the amphora status is not 'DELETED', the amphora is still considered to be in use.

Evacuating a Specific Amphora from a Host

In some cases an amphora needs to be evacuated either because the host is being shutdown for maintenance or as part of a failover. Octavia has a rich amphora API to do that.

First use the amphora API to find the specific amphora. Then, if not already performed, disable scheduling to this host in nova. Lastly, initiate a failover of the specific amphora with the failover command on the amphora API.

Alternatively, a live migration might also work if it happens quick enough for Octavia not to notice a stale amphora (the default configuration is 60s).

1.4.2 octavia-status

CLI interface for Octavia status commands

Synopsis

```
octavia-status <category> <command> [<args>]
```

Description

`octavia-status` is a tool that provides routines for checking the status of a Octavia deployment.

Options

The standard pattern for executing a **`octavia-status`** command is:

```
octavia-status <category> <command> [<args>]
```

Run without arguments to see a list of available command categories:

```
octavia-status
```

Categories are:

- `upgrade`

Detailed descriptions are below:

You can also run with a category argument such as `upgrade` to see a list of all commands in that category:

```
octavia-status upgrade
```

These sections describe the available categories and arguments for **`octavia-status`**.

Upgrade

`octavia-status upgrade check` Performs a release-specific readiness check before restarting services with new code. For example, missing or changed configuration options, incompatible object states, or other conditions that could lead to failures while upgrading.

Return Codes

Return code	Description
0	All upgrade readiness checks passed successfully and there is nothing to do.
1	At least one check encountered an issue and requires further investigation. This is considered a warning but the upgrade may be OK.
2	There was an upgrade status check failure that needs to be investigated. This should be considered something that stops an upgrade.
255	An unexpected error occurred.

History of Checks

4.0.0 (Stein)

- Sample check to be filled in with checks as they are added in Stein.

1.4.3 Load Balancing Service Upgrade Guide

This document outlines steps and notes for operators for reference when upgrading their Load Balancing service from previous versions of OpenStack.

Plan the upgrade

Before jumping right in to the upgrade process, there are a few considerations operators should observe:

- Carefully read the release notes, particularly the upgrade section.
- Upgrades are only supported between sequential releases. For example, upgrading from Pike to Queens is supported while from Pike to Rocky is not.
- It is expected that each Load Balancing provider provides its own upgrade documentation. Please refer to it for upgrade instructions.
- The Load Balancing service builds on top of other OpenStack services, e.g. Compute, Networking, Image and Identify. On a staging environment, upgrade the Load Balancing service and verify it works as expected. For example, a good indicator would be the successful run of *Octavia Tempest tests <https://opendev.org/openstack/octavia-tempest-plugin>*.

Cold upgrade

In a cold upgrade (also known as offline upgrade and non-rolling upgrade), the Load Balancing service is not available because all the control plane services have to be taken down. No data plane disruption should result during the course of upgrading. In the case of the Load Balancing service, it means no downtime nor reconfiguration of service-managed resources (e.g. load balancers, listeners, pools and members).

1. Run the *octavia-status upgrade check* command to validate that Octavia is ready for upgrade.
2. Gracefully stop all Octavia processes. We recommend in this order: Housekeeping, Health manager, API, Worker.
3. Optional: Make a backup of the database.
4. Upgrade all Octavia control plane nodes to the next release. Remember to also upgrade library dependencies (e.g. octavia-lib). If upgrading Octavia from distribution packages, your system package manager is expected to handle this automatically.
5. Verify that all configuration option names are up-to-date with latest Octavia version. For example, pay special attention to deprecated configurations.
6. Run `octavia-db-manage upgrade head` from any Octavia node to upgrade the database and run any corresponding database migrations.
7. Start all Octavia processes.
8. Build a new image and upload it to the Image service. Do not forget to tag the image. We recommend updating images frequently to include latest bug fixes and security issues on installed software (operating system, amphora agent and its dependencies).

Amphorae upgrade

Amphorae upgrade may be required in the advent of API incompatibility between the running amphora agent (old version) and Octavia services (new version). Octavia will automatically recover by failing over amphorae and thus new amphora instances will be running on latest amphora agent code. The drawback in that case is data plane downtime during failover. API breakage is a very rare case, and would be highlighted in the release notes if this scenario occurs.

Upgrade testing

Grenade is an OpenStack test harness project that validates upgrade scenarios between releases. It uses DevStack to initially perform a base OpenStack install and then upgrade to a target version.

Octavia has a Grenade plugin and a CI gate job that validates cold upgrades of an OpenStack deployment with Octavia enabled. The plugin creates load balancing resources and verifies that resources are still working during and after upgrade.

1.5 Operator Reference

1.5.1 Octavia HAProxy Amphora API

Introduction

This document describes the API interface between the reference haproxy driver and its corresponding haproxy-based amphorae.

Octavia reference haproxy amphorae use a web service API for configuration and control. This API should be secured through the use of TLS encryption as well as bi-directional verification of client- and server-side certificates. (The exact process for generating and distributing these certificates should be covered in another document.)

In addition to the web service configuration and control interface, the amphorae may use an HMAC-signed UDP protocol for communicating regular, less- vital information to the controller (ex. statistics updates and health checks). Information on this will also be covered in another document.

If a given loadbalancer is being serviced by multiple haproxy amphorae at the same time, configuration and control actions should be made on all these amphorae at approximately the same time. (Amphorae do not communicate directly with each other, except in an active-standby topology, and then this communication is limited to fail-over protocols.)

Contents

- *Octavia HAProxy Amphora API*
  - *Introduction*
    - *Versioning*
    - *Response codes*
    - *A note about storing state*

  - *API*
    - *Get amphora info*
    - *Get amphora details*
    - *Get interface*
    - *Get all listeners' statuses*
    - *Start or Stop a load balancer*
    - *Delete a listener*
    - *Upload SSL certificate PEM file*
    - *Get SSL certificate md5sum*
    - *Delete SSL certificate PEM file*
    - *Upload load balancer haproxy configuration*
    - *Get loadbalancer haproxy configuration*
    - *Plug VIP*
    - *Plug Network*
    - *Upload SSL server certificate PEM file for Controller Communication*
    - *Upload keepalived configuration*
    - *Start, Stop, or Reload keepalived*
    - *Update the amphora agent configuration*

Versioning

All Octavia APIs (including internal APIs like this one) are versioned. For the purposes of this document, the initial version of this API shall be 1.0.

Response codes

Typical response codes are:

- 200 OK - Operation was completed as requested.
- 201 Created - Operation successfully resulted in the creation / processing of a file.
- 202 Accepted - Command was accepted but is not completed. (Note that this is used for asynchronous processing.)
- 400 Bad Request - API handler was unable to complete request.
- 401 Unauthorized - Authentication of the client certificate failed.
- 404 Not Found - The requested file was not found.
- 500 Internal Server Error - Usually indicates a permissions problem

- 503 Service Unavailable - Usually indicates a change to a listener was attempted during a transition of amphora topology.

A note about storing state

In the below API, it will become apparent that at times the amphora will need to be aware of the state of things (topology-wise, or simply in terms running processes on the amphora). When it comes to storing or gathering this data, we should generally prefer to try to resolve these concerns in the following order. Note also that not every kind of state data will use all of the steps in this list:

1. Get state information by querying running processes (ex. parsing haproxy status page or querying iptables counters, etc.)
2. Get state by consulting on-disk cache generated by querying running processes. (In the case where state information is relatively expensive to collect-- eg. package version listings.)
3. Get state by consulting stored configuration data as sent by the controller. (ex. amphora topology, haproxy configuration or TLS certificate data)
4. Get state by querying a controller API (not described here).

In no case should the amphora assume it ever has direct access to the Octavia database. Also, sensitive data (like TLS certificates) should be stored in a secure way (ex. memory filesystem).

API

Get amphora info

- **URL:** /info
- **Method:** GET
- **URL params:** none
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: JSON formatted listing of several basic amphora data.
- **Error Response:**
  - none

JSON Response attributes:

- *hostname* - amphora hostname
- *uuid* - amphora UUID
- *haproxy_version* - Version of the haproxy installed
- *api_version* - Version of haproxy amphora API in use

Notes: The data in this request is used by the controller for determining the amphora and API version numbers.

It's also worth noting that this is the only API command that doesn't have a version string prepended to it.

Examples:

- Success code 200:

```
{
  'hostname': 'octavia-haproxy-img-00328.local',
  'uuid': '6e2bc8a0-2548-4fb7-a5f0-fb1ef4a696ce',
  'haproxy_version': '1.5.11',
  'api_version': '0.1',
}
```

Get amphora details

- **URL:** /1.0/details
- **Method:** GET
- **URL params:** none
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: JSON formatted listing of various amphora statistics.
- **Error Response:**
  - none

JSON Response attributes:

- *hostname* - amphora hostname
- *uuid* - amphora UUID
- *haproxy_version* - Version of the haproxy installed
- *api_version* - Version of haproxy amphora API/agent in use
- *network_tx* - Current total outbound bandwidth in bytes/sec (30-second snapshot)
- *network_rx* - Current total inbound bandwidth in bytes/sec (30-second snapshot)
- *active* - Boolean (is amphora in an "active" role?)
- *haproxy_count* - Number of running haproxy processes
- *cpu* - list of percent CPU usage broken down into:
  - total
  - user
  - system
  - soft_irq
- *memory* - memory usage in kilobytes broken down into:

  - total
  - free
  - available
  - buffers
  - cached
  - swap_used
  - shared
  - slab
  - committed_as
- *disk* - disk usage in kilobytes for root filesystem, listed as:
  - used
  - available
- *load* - System load (list)
- *topology* - One of SINGLE, ACTIVE-STANDBY, ACTIVE-ACTIVE
- *topology_status* - One of OK, TOPOLOGY-CHANGE
- *listeners* - list of listener UUIDs being serviced by this amphora
- *packages* - list of load-balancing related packages installed with versions (eg. OpenSSL, haproxy, nginx, etc.)

Notes: The data in this request is meant to provide intelligence for an auto-scaling orchestration controller (heat) in order to determine whether additional (or fewer) virtual amphorae are necessary to handle load. As such, we may add additional parameters to the JSON listing above if they prove to be useful for making these decisions.

The data in this request is also used by the controller for determining overall health of the amphora, currently-configured topology and role, etc.

Examples

- Success code 200:

```
{
  'hostname': 'octavia-haproxy-img-00328.local',
  'uuid': '6e2bc8a0-2548-4fb7-a5f0-fb1ef4a696ce',
  'haproxy_version': '1.5.11',
  'api_version': '0.1',
  'networks': {
      'eth0': {
          'network_tx': 3300138,
          'network_rx': 982001, }}
  'active': 'TRUE',
  'haproxy_count': 3,
  'cpu':{
    'total': 0.43,
    'user': 0.30,
```

(continues on next page)

(continued from previous page)

```
    'system': 0.05,
    'soft_irq': 0.08,
  },
  'memory':{
    'total': 4087402,
    'free': 760656,
    'available': 2655901,
    'buffers': 90980,
    'cached': 1830143,
    'swap_used': 943,
    'shared': 105792,
    'slab': 158819,
    'committed_as': 2643480,
  },
  'disk':{
    'used': 1234567,
    'available': 5242880,
  },
  'load': [0.50, 0.45, 0.47],
  'tolopogy': 'SINGLE',
  'topology_status': 'OK',
  'listeners':[
    '02d0da8d-fc65-4bc4-bc46-95cadb2315d2',
    '98e706a7-d22c-422f-9632-499fd83e12c0',
  ],
  'packages':[
    {'haproxy': '1.5.1'},
    {'bash': '4.3.23'},
    {'lighttpd': '1.4.33-1'},
    {'openssl': '1.0.1f'},
    <cut for brevity>
  ],
}
```

Get interface

- **URL:** /1.0/interface/*:ip*
- **Method:** GET
- **URL params:**
  - *:ip* = the ip address to find the interface name
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: OK
    - Content: JSON formatted interface

- **Error Response:**
  - Code: 400
    - Content: Bad IP address version
  - Code: 404
    - Content: Error interface not found for IP address
- **Response:**

OK
eth1

Examples:

- Success code 200:

```
GET URL:
https://octavia-haproxy-img-00328.local/1.0/interface/10.0.0.1

JSON Response:
    {
      'message': 'OK',
      'interface': 'eth1'
    }
```

- Error code 404:

```
GET URL:
https://octavia-haproxy-img-00328.local/1.0/interface/10.5.0.1

JSON Response:
    {
      'message': 'Error interface not found for IP address',
    }
```

- Error code 404:

```
GET URL:
https://octavia-haproxy-img-00328.local/1.0/interface/10.6.0.1.1

JSON Response:
    {
      'message': 'Bad IP address version',
    }
```

Get all listeners' statuses

- **URL:** /1.0/listeners
- **Method:** GET
- **URL params:** none
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: JSON-formatted listing of each listener's status
- **Error Response:**
  - none

JSON Response attributes:

Note that the command will return an array of *all* listeners' statuses. Each listener status contains the following attributes:

- *status* - One of the operational status: ACTIVE, STOPPED, ERROR - future versions might support provisioning status: PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED
- *uuid* - Listener UUID
- *type* - One of: TCP, HTTP, TERMINATED_HTTPS

Notes: Note that this returns a status if: the pid file exists, the stats socket exists, or an haproxy configuration is present (not just if there is a valid haproxy configuration).

Examples

- Success code 200:

```
[{
  'status': 'ACTIVE',
  'uuid': 'e2dfddc0-5b9e-11e4-8ed6-0800200c9a66',
  'type': 'HTTP',
 },
 {
  'status': 'STOPPED',
  'uuid': '19d45130-5b9f-11e4-8ed6-0800200c9a66',
  'type': 'TERMINATED_HTTPS',
 }]
```

Start or Stop a load balancer

- **URL:** /1.0/loadbalancer/*:object_id*/*:action*
- **Method:** PUT
- **URL params:**
    - *:object_id* = Object UUID
    - *:action* = One of: start, stop, reload
- **Data params:** none
- **Success Response:**
    - Code: 202
        - Content: OK
        - *(Also contains preliminary results of attempt to start / stop / soft restart (reload) the haproxy daemon)*
- **Error Response:**
    - Code: 400
        - Content: Invalid request
    - Code: 404
        - Content: Listener Not Found
    - Code: 500
        - Content: Error starting / stopping / reload_config haproxy
        - *(Also contains error output from attempt to start / stop / soft restart (reload) haproxy)*
    - Code: 503
        - Content: Topology transition in progress
- **Response:**

OK
Configuration file is valid
haproxy daemon for 85e2111b-29c4-44be-94f3-e72045805801 started (pid 32428)

Examples:

- Success code 201:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/start

JSON Response:
{
```

(continues on next page)

(continued from previous page)

```
  'message': 'OK',
  'details': 'Configuration file is valid\nhaproxy daemon for 85e2111b-29c4-
↪44be-94f3-e72045805801 started',
}
```

- Error code 400:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/BAD_TEST_DATA

JSON Response:
{
  'message': 'Invalid Request',
  'details': 'Unknown action: BAD_TEST_DATA',
}
```

- Error code 404:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/04bff5c3-5862-4a13-
↪b9e3-9b440d0ed50a/stop

JSON Response:
{
  'message': 'Listener Not Found',
  'details': 'No loadbalancer with UUID: 04bff5c3-5862-4a13-b9e3-9b440d0ed50a
↪',
}
```

- Error code 500:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/stop

Response:
{
  'message': 'Error stopping haproxy',
  'details': 'haproxy process with PID 3352 not found',
}
```

- Error code 503:

```
Response:
{
  'message': 'Topology transition in progress',
}
```

Delete a listener

- **URL:** /1.0/listeners/*:listener*
- **Method:** DELETE
- **URL params:**
  - *:listener* = Listener UUID
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: OK
- **Error Response:**
  - Code: 404
    - Content: Not Found
  - Code: 503
    - Content: Topology transition in progress
- **Response:**

OK

- **Implied actions:**
  - Stop listener
  - Delete IPs, iptables accounting rules, etc. from this amphora if they're no longer in use.
  - Clean up listener configuration directory.
  - Delete listener's SSL certificates
  - Clean up logs (ship final logs to logging destination if configured)
  - Clean up stats socket.

Examples

- Success code 200:

```
DELETE URL:
https://octavia-haproxy-img-00328.local/1.0/listeners/04bff5c3-5862-4a13-b9e3-
↪9b440d0ed50a

JSON Response:
{
  'message': 'OK'
}
```

- Error code 404:

```
DELETE URL:
https://octavia-haproxy-img-00328.local/1.0/listeners/04bff5c3-5862-4a13-b9e3-
↪9b440d0ed50a

JSON Response:
{
  'message': 'Listener Not Found',
  'details': 'No listener with UUID: 04bff5c3-5862-4a13-b9e3-9b440d0ed50a',
}
```

- Error code 503:

```
Response:
{
  'message': 'Topology transition in progress',
}
```

Upload SSL certificate PEM file

- **URL:** /1.0/loadbalancer/*:loadbalancer_id*/certificates/*:filename.pem*
- **Method:** PUT
- **URL params:**
  - *:loadbalancer_id* = Load balancer UUID
  - *:filename* = PEM filename (see notes below for naming convention)
- **Data params:** Certificate data. (PEM file should be a concatenation of unencrypted RSA key, certificate and chain, in that order)
- **Success Response:**
  - Code: 201
    - Content: OK
- **Error Response:**
  - Code: 400
    - Content: No certificate found
  - Code: 400
    - Content: No RSA key found
  - Code: 400
    - Content: Certificate and key do not match
  - Code: 404
    - Content: Not Found
  - Code: 503
    - Content: Topology transition in progress

- **Response:**

OK

Notes: * filename.pem should match the primary CN for which the certificate is valid. All-caps WILDCARD should be used to replace an asterisk in a wildcard certificate (eg. a CN of '*.example.com' should have a filename of 'WILDCARD.example.com.pem'). Filenames must also have the .pem extension. * In order for the new certificate to become effective the haproxy needs to be explicitly restarted

Examples:

- Success code 201:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem
(Put data should contain the certificate information, concatenated as
described above)

JSON Response:
{
  'message': 'OK'
}
```

- Error code 400:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem
(If PUT data does not contain a certificate)

JSON Response:
{
  'message': 'No certificate found'
}
```

- Error code 400:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem
(If PUT data does not contain an RSA key)

JSON Response:
{
  'message': 'No RSA key found'
}
```

- Error code 400:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem
(If the first certificate and the RSA key do not have the same modulus.)

JSON Response:
{
  'message': 'Certificate and key do not match'
}
```

- Error code 404:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem

JSON Response:
{
  'message': 'Listener Not Found',
  'details': 'No loadbalancer with UUID: 04bff5c3-5862-4a13-b9e3-9b440d0ed50a
↪',
}
```

- Error code 503:

```
Response:
{
  'message': 'Topology transition in progress',
}
```

Get SSL certificate md5sum

- **URL:** /1.0/loadbalancer/*:loadbalancer_id*/certificates/*:filename.pem*
- **Method:** GET
- **URL params:**
  - *:loadbalancer_id* = Load balancer UUID
  - *:filename* = PEM filename (see notes below for naming convention)
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: PEM file md5sum
- **Error Response:**
  - Code: 404
    - Content: Not Found

- **Response:**

<certificate PEM file md5 sum>

- **Implied actions:** none

Notes: The md5sum is the sum from the raw certificate data as stored on the amphora (which will usually include the RSA key, certificate and chain concatenated together). Note that we don't return any actual raw certificate data as the controller should already know this information, and unnecessarily disclosing it over the wire from the amphora is a security risk.

Examples:

- Success code 200:

```
JSON response:
{
  'md5sum': 'd8f6629d5e3c6852fa764fb3f04f2ffd',
}
```

- Error code 404:

```
JSON Response:
  {
    'message': 'Listener Not Found',
    'details': 'No loadbalancer with UUID: 04bff5c3-5862-4a13-b9e3-
↪9b440d0ed50a',
  }
```

- Error code 404:

```
JSON Response:
  {
    'message': 'Certificate Not Found',
    'details': 'No certificate with file name: www.example.com.pem',
  }
```

Delete SSL certificate PEM file

- **URL:** /1.0/loadbalancer/*:loadbalancer_id*/certificates/*:filename.pem*
- **Method:** DELETE
- **URL params:**
  - *:loadbalancer_id* = Load balancer UUID
  - *:filename* = PEM filename (see notes below for naming convention)
- **Data params:** none
- **Success Response:**
  - Code: 200

        - Content: OK

- **Error Response:**
    - Code: 404
        - Content: Not found
    - Code: 503
        - Content: Topology transition in progress
- **Implied actions:**
    - Clean up listener configuration directory if it's now empty.

Examples:

- Success code 200:

```
DELETE URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem

JSON Response:
{
  'message': 'OK'
}
```

- Error code 404:

```
 DELETE URL:
 https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/certificates/www.example.com.pem

JSON Response:
     {
       'message': 'Certificate Not Found',
       'details': 'No certificate with file name: www.example.com.pem',
     }
```

- Error code 503:

```
Response:
{
  'message': 'Topology transition in progress',
}
```

Upload load balancer haproxy configuration

- **URL:** /1.0/loadbalancer/*:amphora_id*/*:loadbalancer_id*/haproxy
- **Method:** PUT
- **URL params:**
  - *:loadbalancer_id* = Load Balancer UUID
  - *:amphora_id* = Amphora UUID
- **Data params:** haproxy configuration file for the listener
- **Success Response:**
  - Code: 201
    - Content: OK
- **Error Response:**
  - Code: 400
    - Content: Invalid configuration
    - *(Also includes error output from configuration check command)*
  - Code: 503
    - Content: Topology transition in progress
- **Response:**

OK
Configuration file is valid

- **Implied actions:**
  - Do a syntax check on haproxy configuration file prior to an attempt to run it.
  - Add resources needed for stats, logs, and connectivity

Notes: The uploaded configuration file should be a complete and syntactically-correct haproxy config. The amphora does not have intelligence to generate these itself and has only rudimentary ability to parse certain features out of the configuration file (like bind addresses and ports for purposes of setting up stats, and specially formatted comments meant to indicate pools and members that will be parsed out of the haproxy daemon status interface for tracking health and stats).

Examples:

- Success code 201:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/d459b1c8-54b0-4030-9bec-4f449e73b1ef/85e2111b-29c4-44be-94f3-e72045805801/haproxy
(Upload PUT data should be a raw haproxy.conf file.)

JSON Response:
```

(continues on next page)

(continued from previous page)

```
{
  'message': 'OK'
}
```

- Error code 400:

```
JSON Response:
{
  'message': 'Invalid request',
  'details': '[ALERT] 300/013045 (28236) : parsing [haproxy.cfg:4]: unknown␣
↪keyword 'BAD_LINE' out of section.\n[ALERT] 300/013045 (28236) : Error(s)␣
↪found in configuration file : haproxy.cfg\n[ALERT] 300/013045 (28236) :␣
↪Fatal errors found in configuration.',
}
```

- Error code 503:

```
Response:
{
  'message': 'Topology transition in progress',
}
```

Get loadbalancer haproxy configuration

- **URL:** /1.0/loadbalancer/*:loadbalancer_id*/haproxy
- **Method:** GET
- **URL params:**
  - *:loadbalancer_id* = Load balancer UUID
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: haproxy configuration file for the listener
- **Error Response:**
  - Code: 404
    - Content: Not found
- **Response:**

```
# Config file for 85e2111b-29c4-44be-94f3-e72045805801
(cut for brevity)
```

- **Implied actions:** none

Examples:

- Success code 200:

```
GET URL:
https://octavia-haproxy-img-00328.local/1.0/loadbalancer/85e2111b-29c4-44be-
↪94f3-e72045805801/haproxy

Response is the raw haproxy.cfg:

# Config file for 85e2111b-29c4-44be-94f3-e72045805801
(cut for brevity)
```

- Error code 404:

```
JSON Response:
  {
    'message': 'Loadbalancer Not Found',
    'details': 'No loadbalancer with UUID: 04bff5c3-5862-4a13-b9e3-
↪9b440d0ed50a',
  }
```

Plug VIP

- **URL:** /1.0/plug/vip/*:ip*
- **Method:** Post
- **URL params:**
  - *:ip* = the vip's ip address
- **Data params:**
- *subnet_cidr*: The vip subnet in cidr notation
- *gateway*: The vip subnet gateway address
- *mac_address*: The mac address of the interface to plug
- **Success Response:**
  - Code: 202
    - Content: OK
- **Error Response:** * Code: 400
  - Content: Invalid IP
  - Content: Invalid subnet information
  - Code: 404
    - Content: No suitable network interface found
  - Code: 500
    - Content: Error plugging VIP

    - (Also contains error output from the ip up command)
  - Code: 503
    - Content: Topology transition in progress
- **Response:**

OK
VIP <vip> ip plugged on interface <interface>

- **Implied actions:**
  - Look for an interface marked as down (recently added port)
  - Assign VIP
  - Bring that interface up

Examples:

- Success code 202:

```
POST URL:
https://octavia-haproxy-img-00328.local/1.0/plug/vip/203.0.113.2

JSON POST parameters:
{
  'subnet_cidr': '203.0.113.0/24',
  'gateway': '203.0.113.1',
  'mac_address': '78:31:c1:ce:0b:3c'
}

JSON Response:
    {
      'message': 'OK',
      'details': 'VIP 203.0.113.2 plugged on interface eth1'
    }
```

- Error code 400:

```
JSON Response:
  {
    'message': 'Invalid VIP',
  }
```

- Error code 404:

```
JSON Response:
  {
    'message': 'No suitable network interface found',
  }
```

Plug Network

- **URL:** /1.0/plug/network/
- **Method:** POST
- **URL params:** none
- **Data params:**
- *mac_address*: The mac address of the interface to plug
- **Success Response:**
  - Code: 202
    - Content: OK
- **Error Response:**
  - Code: 404
    - Content: No suitable network interface found
  - Code: 500
    - Content: Error plugging Port
    - (Also contains error output from the ip up command)
  - Code: 503
    - Content: Topology transition in progress
- **Response:**

OK
Plugged interface <interface>

Examples:

- Success code 202:

```
POST URL:
https://octavia-haproxy-img-00328.local/1.0/plug/network/

JSON POST parameters:
{
  'mac_address': '78:31:c1:ce:0b:3c'
}

JSON Response:
    {
      'message': 'OK',
      'details': 'Plugged interface eth1'
    }
```

- Error code 404:

```
JSON Response:
  {
    'message': 'No suitable network interface found',
  }
```

Upload SSL server certificate PEM file for Controller Communication

- **URL:** /1.0/certificate
- **Method:** PUT
- **Data params:** Certificate data. (PEM file should be a concatenation of unencrypted RSA key, certificate and chain, in that order)
- **Success Response:**
  - Code: 202
    - Content: OK
- **Error Response:**
  - Code: 400
    - Content: No certificate found
  - Code: 400
    - Content: No RSA key found
  - Code: 400
    - Content: Certificate and key do not match
- **Response:**

OK

Notes: Since certificates might be valid for a time smaller than the amphora is in existence this add a way to rotate them. Once the certificate is uploaded the agent is being recycled so depending on the implementation the service might not be available for some time.

Examples:

- Success code 202:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/certificate
(Put data should contain the certificate information, concatenated as
described above)

JSON Response:
{
  'message': 'OK'
}
```

- Error code 400:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/certificates
(If PUT data does not contain a certificate)

JSON Response:
{
  'message': 'No certificate found'
}
```

- Error code 400:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/certificate
(If PUT data does not contain an RSA key)

JSON Response:
{
  'message': 'No RSA key found'
}
```

- Error code 400:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/certificate
(If the first certificate and the RSA key do not have the same modulus.)

JSON Response:
{
  'message': 'Certificate and key do not match'
}
```

Upload keepalived configuration

- **URL:** /1.0/vrrp/upload
- **Method:** PUT
- **URL params:** none
- **Data params:** none
- **Success Response:**
  - Code: 200
    - Content: OK
- **Error Response:**
  - Code: 500
    - Content: Failed to upload keepalived configuration.
- **Response:**

OK

Examples:

- Success code 200:

```
PUT URI:
https://octavia-haproxy-img-00328.local/1.0/vrrp/upload

JSON Response:
{
  'message': 'OK'
}
```

Start, Stop, or Reload keepalived

- **URL:** /1.0/vrrp/*:action*
- **Method:** PUT
- **URL params:**
    - *:action* = One of: start, stop, reload
- **Data params:** none
- **Success Response:**
    - Code: 202
        * Content: OK
- **Error Response:**
    - Code: 400
        * Content: Invalid Request
    - Code: 500
        * Content: Failed to start / stop / reload keepalived service:
        * *(Also contains error output from attempt to start / stop / reload keepalived)*
- **Response:**

OK
keepalived started

Examples:

- Success code 202:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/vrrp/start
```

(continues on next page)

(continued from previous page)

```
JSON Response:
{
  'message': 'OK',
  'details': 'keepalived started',
}
```

- Error code: 400

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/vrrp/BAD_TEST_DATA

JSON Response:
{
  'message': 'Invalid Request',
  'details': 'Unknown action: BAD_TEST_DATA',
}
```

- Error code: 500

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/vrrp/stop

JSON Response:
{
  'message': 'Failed to stop keepalived service: keeepalived process with PID␣
↪3352 not found',
  'details': 'keeepalived process with PID 3352 not found',
}
```

Update the amphora agent configuration

- **URL:** /1.0/config
- **Method:** PUT
- **Data params:** A amphora-agent configuration file
- **Success Response:**
  - Code: 202
    - Content: OK
- **Error Response:**
  - Code: 500
    - message: Unable to update amphora-agent configuration.
    - details: *(The exception details)*
- **Response:**

OK

- **Implied actions:**
    - The running amphora-agent configuration file is mutated.

Notes: Only options that are marked mutable in the oslo configuration will be updated.

Examples:

- Success code 202:

```
PUT URL:
https://octavia-haproxy-img-00328.local/1.0/config
(Upload PUT data should be a raw amphora-agent.conf file.)

JSON Response:
{
  'message': 'OK'
}
```

- Error code 500:

```
JSON Response:
{
  'message': 'Unable to update amphora-agent configuration.',
  'details': *(The exception output)*,
}
```

CHAPTER

TWO

OCTAVIA COMMAND LINE INTERFACE

Octavia has an OpenStack Client plugin available as the native Command Line Interface (CLI).

Please see the python-octaviaclient documentation for documentation on installing and using the CLI.

CHAPTER

THREE

OCTAVIA CONFIGURATION

CHAPTER

FOUR

OCTAVIA CONTRIBUTOR

4.1 Contributor Guidelines

4.1.1 So You Want to Contribute...

For general information on contributing to OpenStack, please check out the contributor guide to get started. It covers all the basics that are common to all OpenStack projects: the accounts you need, the basics of interacting with our Gerrit review system, how we communicate as a community, etc.

Below will cover the more project specific information you need to get started with Octavia.

Communication

IRC People working on the Octavia project may be found in the `#openstack-lbaas` channel on the IRC network described in https://docs.openstack.org/contributors/common/irc.html during working hours in their timezone. The channel is logged, so if you ask a question when no one is around, you can check the log to see if it's been answered: http://eavesdrop.openstack.org/irclogs/%23openstack-lbaas/

Weekly Meeting The Octavia team meets weekly on IRC. Please see the OpenStack meetings page for the current meeting details and ICS file: http://eavesdrop.openstack.org/#Octavia_Meeting Meetings are logged: http://eavesdrop.openstack.org/meetings/octavia/

Mailing List We use the openstack-discuss@lists.openstack.org mailing list for asynchronous discussions or to communicate with other OpenStack teams. Use the prefix `[octavia]` in your subject line (it's a high-volume list, so most people use email filters).

More information about the mailing list, including how to subscribe and read the archives, can be found at: http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack-discuss

Virtual Meet-ups From time to time, the Octavia project will have video meetings to address topics not easily covered by the above methods. These are announced well in advance at the weekly meeting and on the mailing list.

Physical Meet-ups The Octavia project usually has a presence at the OpenDev/OpenStack Project Team Gathering that takes place at the beginning of each development cycle. Planning happens on an etherpad whose URL is announced at the weekly meetings and on the mailing list.

Contacting the Core Team

The octavia-core team is an active group of contributors who are responsible for directing and maintaining the Octavia project. As a new contributor, your interaction with this group will be mostly through code reviews, because only members of octavia-core can approve a code change to be merged into the code repository.

Note: Although your contribution will require reviews by members of octavia-core, these aren't the only people whose reviews matter. Anyone with a gerrit account can post reviews, so you can ask other developers you know to review your code ... and you can review theirs. (A good way to learn your way around the codebase is to review other people's patches.)

If you're thinking, "I'm new at this, how can I possibly provide a helpful review?", take a look at How to Review Changes the OpenStack Way.

There are also some Octavia project specific reviewing guidelines in the *Octavia Style Commandments* section of the Octavia Contributor Guide.

You can learn more about the role of core reviewers in the OpenStack governance documentation: https://docs.openstack.org/contributors/common/governance.html#core-reviewer

The membership list of octavia-core is maintained in gerrit: https://review.opendev.org/#/admin/groups/370,members

You can also find the members of the octavia-core team at the Octavia weekly meetings.

New Feature Planning

The Octavia team use both Request For Enhancement (RFE) and Specifications (specs) processes for new features.

RFE When a feature being proposed is easy to understand and will have limited scope, the requester will create an RFE in Storyboard. This is a story that includes the tag **[RFE]** in the subject prefix and has the "**rfe**" tag added to the story.

Once an RFE story is created, a core reviewer or the Project Team Lead (PTL) will approved the RFE by adding the "**rfe-approved**" tag. This signals that the core team understands the feature being proposed and enough detail has been provided to make sure the core team understands the goal of the change.

specs If the new feature is a major change or additon to Octavia that will need a detailed design to be successful, the Octavia team requires a specification (spec) proposal be submitted as a patch.

Octavia specification documents are stored in the /octavia/specs directory in the main Octavia git repository: https://opendev.org/openstack/octavia/src/branch/master/specs This directory includes a template.rst file that includes instructions for creating a new Octavia specification.

These specification documents are then rendered and included in the Project Specifications section of the Octavia Contributor Guide.

Feel free to ask in `#openstack-lbaas` or at the weekly meeting if you have an idea you want to develop and you're not sure whether it requires an RFE or a specification.

The Octavia project observes the OpenStack-wide deadlines, for example, final release of non-client libraries (octavia-lib), final release for client libraries (python-octaviaclient), feature freeze, etc. These are noted and explained on the release schedule for the current development cycle available at: https://releases.openstack.org/

Task Tracking

We track our tasks in Storyboard.

If you're looking for some smaller, easier work item to pick up and get started on, search for the 'low-hanging-fruit' tag.

When you start working on a bug, make sure you assign it to yourself. Otherwise someone else may also start working on it, and we don't want to duplicate efforts. Also, if you find a bug in the code and want to post a fix, make sure you file a bug (and assign it to yourself!) just in case someone else comes across the problem in the meantime.

Reporting a Bug

You found an issue and want to make sure we are aware of it? You can do so on Storyboard.

Please remember to include the following information:

- The version of Octavia and OpenStack you observed the issue in.
- Steps to reproduce.
- Expected behavior.
- Observed behavior.
- The log snippet that contains any error information. Please include the lines directly before the error message(s) as they provide context for the error.

Getting Your Patch Merged

The Octavia project policy is that a patch must have two +2s reviews from the core reviewers before it can be merged.

Patches for Octavia projects must include unit and functional tests that cover the new code. Octavia projects include the "openstack-tox-cover" testing job to help identify test coverage gaps in a patch. This can also be run locally by running "tox -e cover".

In addition, some changes may require a release note. Any patch that changes functionality, adds functionality, or addresses a significant bug should have a release note. Release notes can be created using the "reno" tool by running "reno new <summary-message>".

Keep in mind that the best way to make sure your patches are reviewed in a timely manner is to review other people's patches. We're engaged in a cooperative enterprise here.

Project Team Lead Duties

All common PTL duties are enumerated in the PTL guide.

4.1.2 Octavia Constitution

This document defines the guiding principles that project leadership will be following in creating, improving and maintaining the Octavia project.

Octavia is an OpenStack project

This means we try to run things the same way other "canonized" OpenStack projects operate from a procedural perspective. This is because we hope that Octavia will eventually become a standard part of any OpenStack deployment.

Octavia is as open as OpenStack

Octavia tries to follow the same standards for openness that the OpenStack project also strives to follow: https://wiki.openstack.org/wiki/Open We are committed to open design, development, and community.

Octavia is "free"

We mean that both in the "beer" and in the "speech" sense. That is to say, the reference implementation for Octavia should be made up only of open source components that share the same kind of unencumbered licensing that OpenStack uses.

Note that this does not mean we are against having vendors develop products which can replace some of the components within Octavia. (For example, the Octavia VM images might be replaced by a vendor's proprietary VM image.) Rather, it means that: * The reference implementation should always be open source and unencumbered. * We are typically not interested in making design compromises in order to work with a vendor's proprietary product. If a vendor wants to develop a component for Octavia, then the vendor should bend to Octavia's needs, not the other way around.

Octavia is a load balancer for large operators

That's not to say that small operators can't use it. (In fact, we expect it to work well for small deployments, too.) But what we mean here is that if in creating, improving or maintaining Octavia we somehow make it unable to meet the needs of a typical large operator (or that operator's users), then we have failed.

Octavia follows the best coding and design conventions

For the most part, Octavia tries to follow the coding standards set forth for the OpenStack project in general: https://docs.openstack.org/hacking/latest More specific additional standards can be found in the HACKING.rst file in the same directory as this constitution.

Any exceptions should be well justified and documented. (Comments in or near the breach in coding standards are usually sufficient documentation.)

4.1.3 Octavia Style Commandments

This project was ultimately spawned from work done on the Neutron project. As such, we tend to follow Neutron conventions regarding coding style.

- We follow the OpenStack Style Commandments: https://docs.openstack.org/hacking/latest

Octavia Specific Commandments

- [O316] Change assertTrue(isinstance(A, B)) by optimal assert like assertIsInstance(A, B).
- [O318] Change assert(Not)Equal(A, None) or assert(Not)Equal(None, A) by optimal assert like assertIs(Not)None(A).
- [O319] Validate that debug level logs are not translated.
- [O321] Validate that jsonutils module is used instead of json
- [O322] Don't use author tags
- [O323] Change assertEqual(True, A) or assertEqual(False, A) to the more specific assertTrue(A) or assertFalse(A)
- [O324] Method's default argument shouldn't be mutable
- [O338] Change assertEqual(A in B, True), assertEqual(True, A in B), assertEqual(A in B, False) or assertEqual(False, A in B) to the more specific assertIn/NotIn(A, B)
- [O339] LOG.warn() is not allowed. Use LOG.warning()
- [O340] Don't use xrange()
- [O341] Don't translate logs.
- [0342] Exception messages should be translated
- [O343] Python 3: do not use basestring.
- [O344] Python 3: do not use dict.iteritems.
- [O345] Usage of Python eventlet module not allowed
- [O346] Don't use backslashes for line continuation.
- [O347] Taskflow revert methods must have **kwargs.

Creating Unit Tests

For every new feature, unit tests should be created that both test and (implicitly) document the usage of said feature. If submitting a patch for a bug that had no unit test, a new passing unit test should be added. If a submitted bug fix does have a unit test, be sure to add a new one that fails without the patch and passes with the patch.

Everything is python

Although OpenStack apparently allows either python or C++ code, at this time we don't envision needing anything other than python (and standard, supported open source modules) for anything we intend to do in Octavia.

Idempotency

With as much as is going on inside Octavia, its likely that certain messages and commands will be repeatedly processed. It's important that this doesn't break the functionality of the load balancing service. Therefore, as much as possible, algorithms and interfaces should be made as idempotent as possible.

Centralize intelligence, de-centralize workload

This means that tasks which need to be done relatively infrequently but require either additional knowledge about the state of other components in the Octavia system, advanced logic behind decisions, or otherwise a high degree of intelligence should be done by centralized components (ex. controllers) within the Octavia system. Examples of this might include: * Generating haproxy configuration files * Managing the lifecycle of Octavia amphorae * Moving a loadbalancer instance from one Octavia amphora to another.

On the other hand, tasks done extremely often, or which entail a significant load on the system should be pushed as far out to the most horizontally scalable components as possible. Examples of this might include: * Serving actual client requests to end-users (ie. running haproxy) * Monitoring pool members for failure and sending notifications about this * Processing log files

There will often be a balance that needs to be struck between these two design considerations for any given task for which an algorithm needs to be designed. In considering how to strike this balance, always consider the conditions that will be present in a large operator environment.

Also, as a secondary benefit of centralizing intelligence, minor feature additions and bugfixes can often be accomplished in a large operator environment without having to touch every Octavia amphora running in said environment.

All APIs are versioned

This includes "internal" APIs between Octavia components. Experience coding in the Neutron LBaaS project has taught us that in a large project with many heterogeneous parts, throughout the lifecycle of this project, different parts will evolve at different rates. It is important that these components are allowed to do so without hindering or being hindered by parallel development in other components.

It is also likely that in very large deployments, there might be tens- or hundreds-of-thousands of individual instances of a given component deployed (most likely, the Octavia amphorae). It is unreasonable to expect a large operator to update all of these components at once. Therefore it is likely that for a significant amount of time during a roll-out of a new version, both the old and new versions of a given component must be able to be controlled or otherwise interfaced with by the new components.

Both of the above considerations can be allowed for if we use versioning of APIs where components interact with each other.

Octavia must also keep in mind Neutron LBaaS API versions. Octavia must have the ability to support multiple simultaneous Neutron LBaaS API versions in an effort to allow for Neutron LBaaS API deprecation of URIs. The rationale is that Neutron LBaaS API users should have the ability to transition from one version to the next easily.

Scalability and resilience are as important as functionality

Octavia is meant to be an *operator scale* load balancer. As such, it's usually not enough just to get something working: It also needs to be scalable. For most components, "scalable" implies horizontally scalable.

In any large operational environment, resilience to failures is a necessity. Practically speaking, this means that all components of the system that make up Octavia should be monitored in one way or another, and that where possible automatic recovery from the most common kinds of failures should become a standard feature. Where automatic recovery is not an option, then some form of notification about the failure should be implemented.

Avoid premature optimization

Understand that being "high performance" is often not the same thing as being "scalable." First get the thing to work in an intelligent way. Only worry about making it fast if speed becomes an issue.

Don't repeat yourself

Octavia strives to follow DRY principles. There should be one source of truth, and repetition of code should be avoided.

Security is not an afterthought

The load balancer is often both the most visible public interface to a given user application, but load balancers themselves often have direct access to sensitive components and data within the application environment. Security bugs will happen, but in general we should not approve designs which have known significant security problems, or which could be made more secure by better design.

Octavia should follow industry standards

By "industry standards" we either mean RFCs or well-established best practices. We are generally not interested in defining new standards if a prior open standard already exists. We should also avoid doing things which directly or indirectly contradict established standards.

4.2 Contributor Reference

4.2.1 Provider Driver Development Guide

This document is intended as a guide for developers creating provider drivers for the Octavia API. This guide is intended to be an up to date version of the provider driver specification previously approved.

How Provider Drivers Integrate

Available drivers will be enabled by entries in the Octavia configuration file. Drivers will be loaded via stevedore and Octavia will communicate with drivers through a standard class interface defined below. Most driver functions will be asynchronous to Octavia, and Octavia will provide a library of functions that give drivers a way to update status and statistics. Functions that are synchronous are noted below.

Octavia API functions not listed here will continue to be handled by the Octavia API and will not call into the driver. Examples would be show, list, and quota requests.

In addition, drivers may provide a provider agent that the Octavia driver-agent will launch at start up. This is a long-running process that is intended to support the provider driver.

Driver Entry Points

Provider drivers will be loaded via stevedore. Drivers will have an entry point defined in their setup tools configuration using the Octavia driver namespace "octavia.api.drivers". This entry point name will be used to enable the driver in the Octavia configuration file and as the "provider" parameter users specify when creating a load balancer. An example for the octavia reference driver would be:

```
amphora = octavia.api.drivers.amphora_driver.driver:AmphoraProviderDriver
```

In addition, provider drivers may provide a provider agent also defined by a setup tools entry point. The provider agent namespace is "octavia.driver_agent.provider_agents". This will be called once, at Octavia driver-agent start up, to launch a long-running process. Provider agents must be enabled in the Octavia configuration file. An example provider agent entry point would be:

```
amphora_agent = octavia.api.drivers.amphora_driver.agent:AmphoraProviderAgent
```

Stable Provider Driver Interface

Provider drivers should only access the following Octavia APIs. All other Octavia APIs are not considered stable or safe for provider driver use and may change at any time.

- octavia_lib.api.drivers.data_models
- octavia_lib.api.drivers.driver_lib
- octavia_lib.api.drivers.exceptions
- octavia_lib.api.drivers.provider_base
- octavia_lib.common.constants

Octavia Provider Driver API

Provider drivers will be expected to support the full interface described by the Octavia API, currently v2.0. If a driver does not implement an API function, drivers should fail a request by raising a `NotImplementedError` exception. If a driver implements a function but does not support a particular option passed in by the caller, the driver should raise an `UnsupportedOptionError`.

It is recommended that drivers use the jsonschema package or voluptuous to validate the request against the current driver capabilities.

See the *Exception Model* below for more details.

Note: Driver developers should refer to the official Octavia API reference document for details of the fields and expected outcome of these calls.

Load balancer

Create

Creates a load balancer.

Octavia will pass in the load balancer object with all requested settings.

The load balancer will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the load balancer to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The load balancer python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The provider will be removed as this is used for driver selection.
2. The flavor will be expanded from the provided ID to be the full dictionary representing the flavor metadata.

Load balancer object

As of the writing of this specification the create load balancer object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the resource.
flavor	dict	The flavor keys and values.
availability_zone	dict	The availability zone keys and values.
listeners	list	A list of *Listener objects*.
loadbalancer_id	string	ID of load balancer to create.
name	string	Human-readable name of the resource.
pools	list	A list of *Pool object*.
project_id	string	ID of the project owning this resource.
vip_address	string	The IP address of the Virtual IP (VIP).
vip_network_id	string	The ID of the network for the VIP.
vip_port_id	string	The ID of the VIP port.
vip_qos_policy_id	string	The ID of the qos policy for the VIP.
vip_subnet_id	string	The ID of the subnet for the VIP.

The driver is expected to validate that the driver supports the request and raise an exception if the request cannot be accepted.

VIP port creation

Some provider drivers will want to create the Neutron port for the VIP, and others will want Octavia to create the port instead. In order to support both use cases, the create_vip_port() method will ask provider drivers to create a VIP port. If the driver expects Octavia to create the port, the driver will raise a NotImplementedError exception. Octavia will call this function before calling loadbalancer_create() in order to determine if it should create the VIP port. Octavia will call create_vip_port() with a loadbalancer ID and a partially defined VIP dictionary. Provider drivers that support port creation will create the port and return a fully populated VIP dictionary.

VIP dictionary

Name	Type	Description
project_id	string	ID of the project owning this resource.
vip_address	string	The IP address of the Virtual IP (VIP).
vip_network_id	string	The ID of the network for the VIP.
vip_port_id	string	The ID of the VIP port.
vip_qos_policy_id	string	The ID of the qos policy for the VIP.
vip_subnet_id	string	The ID of the subnet for the VIP.

Creating a Fully Populated Load Balancer

If the "listener" option is specified, the provider driver will iterate through the list and create all of the child objects in addition to creating the load balancer instance.

Delete

Removes an existing load balancer.

Octavia will pass in the load balancer object and cascade boolean as parameters.

The load balancer will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

The API includes an option for cascade delete. When cascade is set to True, the provider driver will delete all child objects of the load balancer.

Failover

Performs a failover of a load balancer.

Octavia will pass in the load balancer ID as a parameter.

The load balancer will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the load balancer to either `ACTIVE` if successfully failed over, or `ERROR` if not failed over.

Failover can mean different things in the context of a provider driver. For example, the Octavia driver replaces the current amphora(s) with another amphora. For another provider driver, failover may mean failing over from an active system to a standby system.

Update

Modifies an existing load balancer using the values supplied in the load balancer object.

Octavia will pass in the original load balancer object which is the baseline for the update, and a load balancer object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update load balancer object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the resource.
loadbalancer_id	string	ID of load balancer to update.
name	string	Human-readable name of the resource.
vip_qos_policy_id	string	The ID of the qos policy for the VIP.

The load balancer will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the load balancer to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):

    def create_vip_port(self, loadbalancer_id, vip_dictionary):
        """Creates a port for a load balancer VIP.

        If the driver supports creating VIP ports, the driver will create a
        VIP port and return the vip_dictionary populated with the vip_port_id.
        If the driver does not support port creation, the driver will raise
        a NotImplementedError.

        :param: loadbalancer_id (string): ID of loadbalancer.
        :param: vip_dictionary (dict): The VIP dictionary.
        :returns: VIP dictionary with vip_port_id.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: The driver does not support creating
          VIP ports.
        """
        raise NotImplementedError()

    def loadbalancer_create(self, loadbalancer):
        """Creates a new load balancer.

        :param loadbalancer (object): The load balancer object.
        :return: Nothing if the create request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: The driver does not support create.
        :raises UnsupportedOptionError: The driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()

    def loadbalancer_delete(self, loadbalancer, cascade=False):
        """Deletes a load balancer.

        :param loadbalancer (object): The load balancer object.
        :param cascade (bool): If True, deletes all child objects (listeners,
          pools, etc.) in addition to the load balancer.
        :return: Nothing if the delete request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        """
        raise NotImplementedError()

    def loadbalancer_failover(self, loadbalancer_id):
        """Performs a fail over of a load balancer.

        :param loadbalancer_id (string): ID of the load balancer to failover.
        :return: Nothing if the failover request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises: NotImplementedError if driver does not support request.
```

(continues on next page)

(continued from previous page)

```
        """
        raise NotImplementedError()

    def loadbalancer_update(self, old_loadbalancer, new_loadbalancer):
        """Updates a load balancer.

        :param old_loadbalancer (object): The baseline load balancer object.
        :param new_loadbalancer (object): The updated load balancer object.
        :return: Nothing if the update request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: The driver does not support request.
        :raises UnsupportedOptionError: The driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()
```

Listener

Create

Creates a listener for a load balancer.

Octavia will pass in the listener object with all requested settings.

The listener will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the listener to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The listener python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer. 2. The default_tls_container_ref will be expanded and provided to the driver in PEM format. 3. The sni_container_refs will be expanded and provided to the driver in PEM format.

Listener object

As of the writing of this specification the create listener object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
client_authentication	string	The TLS client authentication mode. One of the options `NONE`, `OPTIONAL` or `MANDATORY`.
client_ca_tls_container_data	string	A PEM encoded certificate.
client_ca_tls_container_ref	string	The reference to the secrets container.
client_crl_container_data	string	A PEM encoded CRL file.
client_crl_container_ref	string	The reference to the secrets container.
connection_limit	int	The max number of connections permitted for this listener. Default is -1, which is infinite connections.
default_pool	object	A *Pool object*.
default_pool_id	string	The ID of the pool used by the listener if no L7 policies match.
default_tls_container_data	dict	A *TLS container* dict.
default_tls_container_refs	string	The reference to the secrets container.
description	string	A human-readable description for the listener.
insert_headers	dict	A dictionary of optional headers to insert into the request before it is sent to the backend member. See *Supported HTTP Header Insertions*. Keys and values are specified as strings.
l7policies	list	A list of *L7policy objects*.
listener_id	string	ID of listener to create.
loadbalancer_id	string	ID of load balancer.
name	string	Human-readable name of the listener.
project_id	string	ID of the project owning this resource.
protocol	string	Protocol type: One of HTTP, HTTPS, TCP, or TERMINATED_HTTPS.
protocol_port	int	Protocol port number.
sni_container_data	list	A list of *TLS container* dict.
sni_container_refs	list	A list of references to the SNI secrets containers.
timeout_client_data	int	Frontend client inactivity timeout in milliseconds.
timeout_member_connect	int	Backend member connection timeout in milliseconds.
timeout_member_data	int	Backend member inactivity timeout in milliseconds.
timeout_tcp_inspect	int	Time, in milliseconds, to wait for additional TCP packets for content inspection.
allowed_cidrs	list	List of IPv4 or IPv6 CIDRs.

As of the writing of this specification the TLS container dictionary contains the following:

Key	Type	Description
certificate	string	The PEM encoded certificate.
intermediates	List	A list of intermediate PEM certificates.
passphrase	string	The private_key passphrase.
primary_cn	string	The primary common name of the certificate.
private_key	string	The PEM encoded private key.

As of the writing of this specification the Supported HTTP Header Insertions are:

Key	Type	Description
X-Forwarded-For	bool	When True a X-Forwarded-For header is inserted into the request to the backend member that specifies the client IP address.
X-Forwarded-Port	int	A X-Forwarded-Port header is inserted into the request to the backend member that specifies the integer provided. Typically this is used to indicate the port the client connected to on the load balancer.
X-Forwarded-Proto	bool	A X-Forwarded-Proto header is inserted into the end of request to the backend member. HTTP for the HTTP listener protocol type, HTTPS for the TERMINATED_HTTPS listener protocol type.
X-SSL-Client-Verify	string	When `"true"` a `X-SSL-Client-Verify` header is inserted into the request to the backend `member` that contains 0 if the client authentication was successful, or an result error number greater than 0 that align to the openssl veryify error codes.
X-SSL-Client-Has-Cert	string	When `"true"` a `X-SSL-Client-Has-Cert` header is inserted into the request to the backend `member` that is ”true” if a client authentication certificate was presented, and ”false” if not. Does not indicate validity.
X-SSL-Client-DN	string	When `"true"` a `X-SSL-Client-DN` header is inserted into the request to the backend `member` that contains the full Distinguished Name of the certificate presented by the client.
X-SSL-Client-CN	string	When `"true"` a `X-SSL-Client-CN` header is inserted into the request to the backend `member` that contains the Common Name from the full Distinguished Name of the certificate presented by the client.
X-SSL-Issuer	string	When `"true"` a `X-SSL-Issuer` header is inserted into the request to the backend `member` that contains the full Distinguished Name of the client certificate issuer.
X-SSL-Client-SHA1	string	When `"true"` a `X-SSL-Client-SHA1` header is inserted into the request to the backend `member` that contains the SHA-1 fingerprint of the certificate presented by the client in hex string format.
X-SSL-Client-Not-Before	string	When `"true"` a `X-SSL-Client-Not-Before` header is inserted into the request to the backend `member` that contains the start date presented by the client as a formatted string YYMMDDhhmmss[Z].
X-SSL-Client-Not-After	string	When `"true"` a `X-SSL-Client-Not-After` header is inserted into the request to the backend `member` that contains the end date presented by the client as a formatted string YYMMDDhhmmss[Z].

Creating a Fully Populated Listener

If the "default_pool" or "l7policies" option is specified, the provider driver will create all of the child objects in addition to creating the listener instance.

Delete

Deletes an existing listener.

Octavia will pass the listener object as a parameter.

The listener will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

Update

Modifies an existing listener using the values supplied in the listener object.

Octavia will pass in the original listener object which is the baseline for the update, and a listener object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update listener object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
client_authentication	string	The TLS client authentication mode. One of the options `NONE`, `OPTIONAL` or `MANDATORY`.
client_ca_tls_container_data	string	A PEM encoded certificate.
client_ca_tls_container_ref	string	The reference to the secrets container.
client_crl_container_data	string	A PEM encoded CRL file.
client_crl_container_ref	string	The reference to the secrets container.
connection_limit	int	The max number of connections permitted for this listener. Default is -1, which is infinite connections.
default_pool_id	string	The ID of the pool used by the listener if no L7 policies match.
default_tls_container_data	dict	A *TLS container* dict.
default_tls_container_refs	string	The reference to the secrets container.
description	string	A human-readable description for the listener.
insert_headers	dict	A dictionary of optional headers to insert into the request before it is sent to the backend member. See *Supported HTTP Header Insertions*. Keys and values are specified as strings.
listener_id	string	ID of listener to update.
name	string	Human-readable name of the listener.
sni_container_data	list	A list of *TLS container* dict.
sni_container_refs	list	A list of references to the SNI secrets containers.
timeout_client_data	int	Frontend client inactivity timeout in milliseconds.
timeout_member_connect	int	Backend member connection timeout in milliseconds.
timeout_member_data	int	Backend member inactivity timeout in milliseconds.
timeout_tcp_inspect	int	Time, in milliseconds, to wait for additional TCP packets for content inspection.
allowed_cidrs	list | List of IPv4 or IPv6 CIDRs.	

The listener will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the listener to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def listener_create(self, listener):
      """Creates a new listener.

      :param listener (object): The listener object.
```

(continues on next page)

(continued from previous page)

```
        :return: Nothing if the create request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
      raise NotImplementedError()

    def listener_delete(self, listener):
        """Deletes a listener.

        :param listener (object): The listener object.
        :return: Nothing if the delete request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        """
        raise NotImplementedError()

    def listener_update(self, old_listener, new_listener):
        """Updates a listener.

        :param old_listener (object): The baseline listener object.
        :param new_listener (object): The updated listener object.
        :return: Nothing if the update request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()
```

Pool

Create

Creates a pool for a load balancer.

Octavia will pass in the pool object with all requested settings.

The pool will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the pool to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The pool python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.

Pool object

As of the writing of this specification the create pool object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
ca_tls_container_data	string	A PEM encoded certificate.
ca_tls_container_ref	string	The reference to the secrets container.
crl_container_data	string	A PEM encoded CRL file.
crl_container_ref	string	The reference to the secrets container.
description	string	A human-readable description for the pool.
healthmonitor	object	A *Healthmonitor object*.
lb_algorithm	string	Load balancing algorithm: One of ROUND_ROBIN, LEAST_CONNECTIONS, SOURCE_IP or SOURCE_IP_PORT.
loadbalancer_id	string	ID of load balancer.
listener_id	string	ID of listener.
members	list	A list of *Member objects*.
name	string	Human-readable name of the pool.
pool_id	string	ID of pool to create.
project_id	string	ID of the project owning this resource.
protocol	string	Protocol type: One of HTTP, HTTPS, PROXY, or TCP.
session_persistence	dict	Defines session persistence as one of {'type': <'HTTP_COOKIE' | 'SOURCE_IP'>} OR {'type': 'APP_COOKIE', 'cookie_name': <cookie_name>}
tls_container_data	dict	A *TLS container* dict.
tls_container_ref	string	The reference to the secrets container.
tls_enabled	bool	True when backend re-encryption is enabled.

Delete

Removes an existing pool and all of its members.

Octavia will pass the pool object as a parameter.

The pool will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

Update

Modifies an existing pool using the values supplied in the pool object.

Octavia will pass in the original pool object which is the baseline for the update, and a pool object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update pool object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
ca_tls_container_data	string	A PEM encoded certificate.
ca_tls_container_ref	string	The reference to the secrets container.
crl_container_data	string	A PEM encoded CRL file.
crl_container_ref	string	The reference to the secrets container.
description	string	A human-readable description for the pool.
lb_algorithm	string	Load balancing algorithm: One of ROUND_ROBIN, LEAST_CONNECTIONS, or SOURCE_IP.
name	string	Human-readable name of the pool.
pool_id	string	ID of pool to update.
session_persistence	dict	Defines session persistence as one of {'type': <'HTTP_COOKIE' | 'SOURCE_IP'>} OR {'type': 'APP_COOKIE', 'cookie_name': <cookie_name>}
tls_container_data	dict	A *TLS container* dict.
tls_container_ref	string	The reference to the secrets container.
tls_enabled	bool	True when backend re-encryption is enabled.

The pool will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the pool to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def pool_create(self, pool):
      """Creates a new pool.

      :param pool (object): The pool object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def pool_delete(self, pool):
      """Deletes a pool and its members.

      :param pool (object): The pool object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      """
      raise NotImplementedError()
```

(continues on next page)

(continued from previous page)

```
def pool_update(self, old_pool, new_pool):
    """Updates a pool.

    :param old_pool (object): The baseline pool object.
    :param new_pool (object): The updated pool object.
    :return: Nothing if the create request was accepted.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: if driver does not support request.
    :raises UnsupportedOptionError: if driver does not
      support one of the configuration options.
    """
    raise NotImplementedError()
```

Member

Create

Creates a member for a pool.

Octavia will pass in the member object with all requested settings.

The member will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the member to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The member python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The project_id will be removed, if present, as this field is now deprecated. The member will inherit the project_id from the parent load balancer.

Member object

As of the writing of this specification the create member object may contain the following:

Name	Type	Description
address	string	The IP address of the backend member to receive traffic from the load balancer.
admin_state_up	bool	Admin state: True if up, False if down.
backup	bool	Is the member a backup? Backup members only receive traffic when all non-backup members are down.
member_id	string	ID of member to create.
monitor_address	string	An alternate IP address used for health monitoring a backend member.
monitor_port	int	An alternate protocol port used for health monitoring a backend member.
name	string	Human-readable name of the member.
pool_id	string	ID of pool.
project_id	string	ID of the project owning this resource.
protocol_port	int	The port on which the backend member listens for traffic.
subnet_id	string	Subnet ID.
weight	int	The weight of a member determines the portion of requests or connections it services compared to the other members of the pool. For example, a member with a weight of 10 receives five times as many requests as a member with a weight of 2. A value of 0 means the member does not receive new connections but continues to service existing connections. A valid value is from 0 to 256. Default is 1.

Delete

Removes a pool member.

Octavia will pass the member object as a parameter.

The member will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

Update

Modifies an existing member using the values supplied in the listener object.

Octavia will pass in the original member object which is the baseline for the update, and a member object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update member object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
backup	bool	Is the member a backup? Backup members only receive traffic when all non-backup members are down.
member_id	string	ID of member to update.
monitor_address	string	An alternate IP address used for health monitoring a backend member.
monitor_port	int	An alternate protocol port used for health monitoring a backend member.
name	string	Human-readable name of the member.
weight	int	The weight of a member determines the portion of requests or connections it services compared to the other members of the pool. For example, a member with a weight of 10 receives five times as many requests as a member with a weight of 2. A value of 0 means the member does not receive new connections but continues to service existing connections. A valid value is from 0 to 256. Default is 1.

The member will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the member to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Batch Update

Set the state of members for a pool in one API call. This may include creating new members, deleting old members, and updating existing members. Existing members are matched based on address/port combination.

For example, assume a pool currently has two members. These members have the following address/port combinations: '192.0.2.15:80' and '192.0.2.16:80'. Now assume a PUT request is made that includes members with address/port combinations: '192.0.2.16:80' and '192.0.2.17:80'. The member '192.0.2.15:80' will be deleted because it was not in the request. The member '192.0.2.16:80' will be updated to match the request data for that member, because it was matched. The member '192.0.2.17:80' will be created, because no such member existed.

The members will be in the `PENDING_CREATE`, `PENDING_UPDATE`, or `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the members to either `ACTIVE` or `DELETED` if successfully updated, or `ERROR` if the update was not successful.

The batch update method will supply a list of *Member objects*. Existing members not in this list should be deleted, existing members in the list should be updated, and members in the list that do not already exist should be created.

Abstract class definition

```
class Driver(object):
  def member_create(self, member):
      """Creates a new member for a pool.

      :param member (object): The member object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
  raise NotImplementedError()

  def member_delete(self, member):

      """Deletes a pool member.

      :param member (object): The member object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      """
      raise NotImplementedError()

  def member_update(self, old_member, new_member):

      """Updates a pool member.

      :param old_member (object): The baseline member object.
      :param new_member (object): The updated member object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def member_batch_update(self, pool_id, members):
      """Creates, updates, or deletes a set of pool members.

      :param pool_id (string): The id of the pool to update.
      :param members (list): List of member objects.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()
```

Health Monitor

Create

Creates a health monitor on a pool.

Octavia will pass in the health monitor object with all requested settings.

The health monitor will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the health monitor to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The healthmonitor python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.

Healthmonitor object

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
delay	int	The interval, in seconds, between health checks.
domain_name	string	The domain name to be passed in the host header for health monitor checks.
expected_codes	string	The expected HTTP status codes to get from a successful health check. This may be a single value, a list, or a range.
healthmonitor_id	string	ID of health monitor to create.
http_method	string	The HTTP method that the health monitor uses for requests. One of CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT, or TRACE.
http_version	float	The HTTP version to use for health monitor connections. One of '1.0' or '1.1'. Defaults to '1.0'.
max_retries	int	The number of successful checks before changing the operating status of the member to ONLINE.
max_retries_down	int	The number of allowed check failures before changing the operating status of the member to ERROR. A valid value is from 1 to 10.
name	string	Human-readable name of the monitor.
pool_id	string	The pool to monitor.
project_id	string	ID of the project owning this resource.
timeout	int	The time, in seconds, after which a health check times out. This value must be less than the delay value.
type	string	The type of health monitor. One of HTTP, HTTPS, PING, SCTP, TCP, TLS-HELLO or UDP-CONNECT.
url_path	string	The HTTP URL path of the request sent by the monitor to test the health of a backend member. Must be a string that begins with a forward slash (/).

Delete

Deletes an existing health monitor.

Octavia will pass in the health monitor object as a parameter.

The health monitor will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

Update

Modifies an existing health monitor using the values supplied in the health monitor object.

Octavia will pass in the original health monitor object which is the baseline for the update, and a health monitor object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update health monitor object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
delay	int	The interval, in seconds, between health checks.
domain_name	string	The domain name to be passed in the host header for health monitor checks.
expected_codes	string	The expected HTTP status codes to get from a successful health check. This may be a single value, a list, or a range.
healthmonitor_id	string	ID of health monitor to create.
http_method	string	The HTTP method that the health monitor uses for requests. One of CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT, or TRACE.
http_version	float	The HTTP version to use for health monitor connections. One of ’1.0’ or ’1.1’. Defaults to ’1.0’.
max_retries	int	The number of successful checks before changing the operating status of the member to ONLINE.
max_retries_down	int	The number of allowed check failures before changing the operating status of the member to ERROR. A valid value is from 1 to 10.
name	string	Human-readable name of the monitor.
timeout	int	The time, in seconds, after which a health check times out. This value must be less than the delay value.
url_path	string	The HTTP URL path of the request sent by the monitor to test the health of a backend member. Must be a string that begins with a forward slash (/).

The health monitor will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the health monitor to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def health_monitor_create(self, healthmonitor):
      """Creates a new health monitor.

      :param healthmonitor (object): The health monitor object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def health_monitor_delete(self, healthmonitor):
      """Deletes a healthmonitor_id.

      :param healthmonitor (object): The health monitor object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      """
      raise NotImplementedError()

  def health_monitor_update(self, old_healthmonitor, new_healthmonitor):
      """Updates a health monitor.

      :param old_healthmonitor (object): The baseline health monitor
        object.
      :param new_healthmonitor (object): The updated health monitor object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()
```

L7 Policy

Create

Creates an L7 policy.

Octavia will pass in the L7 policy object with all requested settings.

The L7 policy will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the L7 policy to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The l7policy python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The project_id will be removed, if present, as this field is now deprecated. The l7policy will inherit the project_id from the parent load balancer.

L7policy object

As of the writing of this specification the create l7policy object may contain the following:

Name	Type	Description
action	string	The L7 policy action. One of REDIRECT_TO_POOL, REDIRECT_TO_URL, or REJECT.
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the L7 policy.
l7policy_id	string	The ID of the L7 policy.
listener_id	string	The ID of the listener.
name	string	Human-readable name of the L7 policy.
position	int	The position of this policy on the listener. Positions start at 1.
project_id	string	ID of the project owning this resource.
redirect_http_code	int	The HTTP status code to be returned on a redirect policy.
redirect_pool_id	string	Requests matching this policy will be redirected to the pool with this ID. Only valid if action is REDIRECT_TO_POOL.
redirect_prefix	string	Requests matching this policy will be redirected to this Prefix URL. Only valid if `action` is `REDIRECT_PREFIX`.
redirect_url	string	Requests matching this policy will be redirected to this URL. Only valid if action is REDIRECT_TO_URL.
rules	list	A list of l7rule objects.

Creating a Fully Populated L7 policy

If the "rules" option is specified, the provider driver will create all of the child objects in addition to creating the L7 policy instance.

Delete

Deletes an existing L7 policy.

Octavia will pass in the L7 policy object as a parameter.

The l7policy will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

Update

Modifies an existing L7 policy using the values supplied in the l7policy object.

Octavia will pass in the original L7 policy object which is the baseline for the update, and an L7 policy object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update L7 policy object may contain the following:

Name	Type	Description
action	string	The L7 policy action. One of REDIRECT_TO_POOL, REDIRECT_TO_URL, or REJECT.
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the L7 policy.
l7policy_id	string	The ID of the L7 policy.
name	string	Human-readable name of the L7 policy.
position	int	The position of this policy on the listener. Positions start at 1.
redirect_http_code	int	The HTTP status code to be returned on a redirect policy.
redirect_pool_id	string	Requests matching this policy will be redirected to the pool with this ID. Only valid if action is REDIRECT_TO_POOL.
redirect_prefix	string	Requests matching this policy will be redirected to this Prefix URL. Only valid if `action` is `REDIRECT_PREFIX`.
redirect_url	string	Requests matching this policy will be redirected to this URL. Only valid if action is REDIRECT_TO_URL.

The L7 policy will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the L7 policy to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def l7policy_create(self, l7policy):
      """Creates a new L7 policy.

      :param l7policy (object): The l7policy object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def l7policy_delete(self, l7policy):
```

(continues on next page)

(continued from previous page)

```
    """Deletes an L7 policy.

    :param l7policy (object): The l7policy object.
    :return: Nothing if the delete request was accepted.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: if driver does not support request.
    """
    raise NotImplementedError()

def l7policy_update(self, old_l7policy, new_l7policy):
    """Updates an L7 policy.

    :param old_l7policy (object): The baseline l7policy object.
    :param new_l7policy (object): The updated l7policy object.
    :return: Nothing if the update request was accepted.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: if driver does not support request.
    :raises UnsupportedOptionError: if driver does not
      support one of the configuration options.
    """
    raise NotImplementedError()
```

L7 Rule

Create

Creates a new L7 rule for an existing L7 policy.

Octavia will pass in the L7 rule object with all requested settings.

The L7 rule will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the L7 rule to either `ACTIVE` if successfully created, or `ERROR` if not created.

The Octavia API will accept and do basic API validation of the create request from the user. The l7rule python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.

L7rule object

As of the writing of this specification the create l7rule object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
compare_type	string	The comparison type for the L7 rule. One of CONTAINS, ENDS_WITH, EQUAL_TO, REGEX, or STARTS_WITH.
invert	bool	When True the logic of the rule is inverted. For example, with invert True, equal to would become not equal to.
key	string	The key to use for the comparison. For example, the name of the cookie to evaluate.
l7policy_id	string	The ID of the L7 policy.
l7rule_id	string	The ID of the L7 rule.
project_id	string	ID of the project owning this resource.
type	string	The L7 rule type. One of COOKIE, FILE_TYPE, HEADER, HOST_NAME, or PATH.
value	string	The value to use for the comparison. For example, the file type to compare.

Delete

Deletes an existing L7 rule.

Octavia will pass in the L7 rule object as a parameter.

The L7 rule will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

Update

Modifies an existing L7 rule using the values supplied in the l7rule object.

Octavia will pass in the original L7 rule object which is the baseline for the update, and an L7 rule object with the fields to be updated. Fields not updated by the user will contain "Unset" as defined in the data model.

As of the writing of this specification the update L7 rule object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
compare_type	string	The comparison type for the L7 rule. One of CONTAINS, ENDS_WITH, EQUAL_TO, REGEX, or STARTS_WITH.
invert	bool	When True the logic of the rule is inverted. For example, with invert True, equal to would become not equal to.
key	string	The key to use for the comparison. For example, the name of the cookie to evaluate.
l7rule_id	string	The ID of the L7 rule.
type	string	The L7 rule type. One of COOKIE, FILE_TYPE, HEADER, HOST_NAME, or PATH.
value	string	The value to use for the comparison. For example, the file type to compare.

The L7 rule will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the L7 rule to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```python
class Driver(object):
    def l7rule_create(self, l7rule):

        """Creates a new L7 rule.

        :param l7rule (object): The L7 rule object.
        :return: Nothing if the create request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()

    def l7rule_delete(self, l7rule):

        """Deletes an L7 rule.

        :param l7rule (object): The L7 rule object.
        :return: Nothing if the delete request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        """
        raise NotImplementedError()

    def l7rule_update(self, old_l7rule, new_l7rule):

        """Updates an L7 rule.

        :param old_l7rule (object): The baseline L7 rule object.
        :param new_l7rule (object): The updated L7 rule object.
        :return: Nothing if the update request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()
```

Flavor

Octavia flavors are defined in a separate flavor specification. Support for flavors will be provided through two provider driver interfaces, one to query supported flavor metadata keys and another to validate that a flavor is supported. Both functions are synchronous.

get_supported_flavor_metadata

Retrieves a dictionary of supported flavor keys and their description. For example:

```
{"topology": "The load balancer topology for the flavor. One of: SINGLE,␣
↪ACTIVE_STANDBY",
 "compute_flavor": "The compute driver flavor to use for the load balancer␣
↪instances"}
```

validate_flavor

Validates that the driver supports the flavor metadata dictionary.

The validate_flavor method will be passed a flavor metadata dictionary that the driver will validate. This is used when an operator uploads a new flavor that applies to the driver.

The validate_flavor method will either return or raise a `UnsupportedOptionError` exception.

Following are interface definitions for flavor support:

```
def get_supported_flavor_metadata():
    """Returns a dictionary of flavor metadata keys supported by this driver.

    The returned dictionary will include key/value pairs, 'name' and
    'description.'

    :returns: The flavor metadata dictionary
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: The driver does not support flavors.
    """
    raise NotImplementedError()
```

```
def validate_flavor(flavor_metadata):
    """Validates if driver can support flavor as defined in flavor_metadata.

    :param flavor_metadata (dict): Dictionary with flavor metadata.
    :return: Nothing if the flavor is valid and supported.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: The driver does not support flavors.
    :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
    """
    raise NotImplementedError()
```

Availability Zone

Octavia availability zones have no explicit spec, but are modeled closely after the existing flavor specification. Support for availability_zones will be provided through two provider driver interfaces, one to query supported availability zone metadata keys and another to validate that an availability zone is supported. Both functions are synchronous.

get_supported_availability_zone_metadata

Retrieves a dictionary of supported availability zone keys and their description. For example:

```
{"compute_zone": "The compute availability zone to use for this loadbalancer.
↪",
 "management_network": "The management network ID for the loadbalancer.",
 "valid_vip_networks": "List of network IDs that are allowed for VIP use.␣
↪This overrides/replaces the list of allowed networks configured in `octavia.
↪conf`."}
```

validate_availability_zone

Validates that the driver supports the availability zone metadata dictionary.

The validate_availability_zone method will be passed an availability zone metadata dictionary that the driver will validate. This is used when an operator uploads a new availability zone that applies to the driver.

The validate_availability_zone method will either return or raise a `UnsupportedOptionError` exception.

Following are interface definitions for availability zone support:

```
def get_supported_availability_zone_metadata():
    """Returns a dict of supported availability zone metadata keys.

      The returned dictionary will include key/value pairs, 'name' and
      'description.'

      :returns: The availability zone metadata dictionary
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: The driver does not support AZs.
      """
    raise NotImplementedError()
```

```
def validate_availability_zone(availability_zone_metadata):
    """Validates if driver can support the availability zone.

      :param availability_zone_metadata: Dictionary with az metadata.
      :type availability_zone_metadata: dict
      :return: Nothing if the availability zone is valid and supported.
      :raises DriverError: An unexpected error occurred in the driver.
```

(continues on next page)

(continued from previous page)

```
        :raises NotImplementedError: The driver does not support availability
          zones.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
    raise NotImplementedError()
```

Exception Model

DriverError

This is a catch all exception that drivers can return if there is an unexpected error. An example might be a delete call for a load balancer the driver does not recognize. This exception includes two strings: The user fault string and the optional operator fault string. The user fault string, "user_fault_string", will be provided to the API requester. The operator fault string, "operator_fault_string", will be logged in the Octavia API log file for the operator to use when debugging.

```
class DriverError(Exception):
    user_fault_string = _("An unknown driver error occurred.")
    operator_fault_string = _("An unknown driver error occurred.")

    def __init__(self, *args, **kwargs):
        self.user_fault_string = kwargs.pop('user_fault_string',
                                            self.user_fault_string)
        self.operator_fault_string = kwargs.pop('operator_fault_string',
                                                self.operator_fault_string)

        super(DriverError, self).__init__(*args, **kwargs)
```

NotImplementedError

Driver implementations may not support all operations, and are free to reject a request. If the driver does not implement an API function, the driver will raise a NotImplementedError exception.

```
class NotImplementedError(Exception):
    user_fault_string = _("A feature is not implemented by this driver.")
    operator_fault_string = _("A feature is not implemented by this driver.")

    def __init__(self, *args, **kwargs):
        self.user_fault_string = kwargs.pop('user_fault_string',
                                            self.user_fault_string)
        self.operator_fault_string = kwargs.pop('operator_fault_string',
                                                self.operator_fault_string)

        super(NotImplementedError, self).__init__(*args, **kwargs)
```

UnsupportedOptionError

Provider drivers will validate that they can complete the request -- that all options are supported by the driver. If the request fails validation, drivers will raise an UnsupportedOptionError exception. For example, if a driver does not support a flavor passed as an option to load balancer create(), the driver will raise an UnsupportedOptionError and include a message parameter providing an explanation of the failure.

```
class UnsupportedOptionError(Exception):
    user_fault_string = _("A specified option is not supported by this driver.
↪")
    operator_fault_string = _("A specified option is not supported by this␣
↪driver.")

    def __init__(self, *args, **kwargs):
        self.user_fault_string = kwargs.pop('user_fault_string',
                                            self.user_fault_string)
        self.operator_fault_string = kwargs.pop('operator_fault_string',
                                                self.operator_fault_string)

        super(UnsupportedOptionError, self).__init__(*args, **kwargs)
```

Driver Support Library

Provider drivers need support for updating provisioning status, operating status, and statistics. Drivers will not directly use database operations, and instead will callback to octavia-lib using a new API.

Warning: The methods listed here are the only callable methods for drivers. All other interfaces are not considered stable or safe for drivers to access. See *Stable Provider Driver Interface* for a list of acceptable APIs for provider driver use.

Warning: This library is interim and will be removed when the driver support endpoint is made available. At which point drivers will not import any code from octavia-lib.

Update Provisioning and Operating Status API

The update status API defined below can be used by provider drivers to update the provisioning and/or operating status of Octavia resources (load balancer, listener, pool, member, health monitor, L7 policy, or L7 rule).

For the following status API, valid values for provisioning status and operating status parameters are as defined by Octavia status codes. If an existing object is not included in the input parameter, the status remains unchanged.

Note: If the driver-agent exceeds its configured *status_max_processes* this call may block while it waits for a status process slot to become available. The operator will be notified if the driver-agent approaches or reaches the configured limit.

---

provisioning_status: status associated with lifecycle of the resource. See Octavia Provisioning Status Codes.

operating_status: the observed status of the resource. See Octavia Operating Status Codes.

The dictionary takes this form:

```
{ "loadbalancers": [{"id": "123",
                     "provisioning_status": "ACTIVE",
                     "operating_status": "ONLINE"},...],
  "healthmonitors": [],
  "l7policies": [],
  "l7rules": [],
  "listeners": [],
  "members": [],
  "pools": []
}
```

```
def update_loadbalancer_status(status):
    """Update load balancer status.

    :param status (dict): dictionary defining the provisioning status and
        operating status for load balancer objects, including pools,
        members, listeners, L7 policies, and L7 rules.
    :raises: UpdateStatusError
    :returns: None
    """
```

Update Statistics API

Provider drivers can update statistics for listeners using the following API. Similar to the status function above, a single dictionary with multiple listener statistics is used to update statistics in a single call. If an existing listener is not included, the statistics that object remain unchanged.

Note: If the driver-agent exceeds its configured *stats_max_processes* this call may block while it waits for a stats process slot to become available. The operator will be notified if the driver-agent approaches or reaches the configured limit.

The general form of the input dictionary is a list of listener statistics:

```
{ "listeners": [{"id": "123",
                 "active_connections": 12,
                 "bytes_in": 238908,
                 "bytes_out": 290234,
                 "request_errors": 0,
                 "total_connections": 3530},...]
}
```

```
def update_listener_statistics(statistics):
    """Update listener statistics.

    :param statistics (dict): Statistics for listeners:
          id (string): ID of the listener.
          active_connections (int): Number of currently active connections.
          bytes_in (int): Total bytes received.
          bytes_out (int): Total bytes sent.
          request_errors (int): Total requests not fulfilled.
          total_connections (int): The total connections handled.
    :raises: UpdateStatisticsError
    :returns: None
    """
```

Get Resource Support

Provider drivers may need to get information about an Octavia resource. As an example of its use, a provider driver may need to sync with Octavia, and therefore need to fetch all of the Octavia resources it is responsible for managing. Provider drivers can use the existing Octavia API to get these resources. See the Octavia API Reference.

API Exception Model

The driver support API will include exceptions: two API groups:

- UpdateStatusError
- UpdateStatisticsError
- DriverAgentNotFound
- DriverAgentTimeout

Each exception class will include a message field that describes the error and references to the failed record if available.

```
class UpdateStatusError(Exception):
    fault_string = _("The status update had an unknown error.")
    status_object = None
    status_object_id = None
    status_record = None

    def __init__(self, *args, **kwargs):
        self.fault_string = kwargs.pop('fault_string',
                                       self.fault_string)
        self.status_object = kwargs.pop('status_object', None)
        self.status_object_id = kwargs.pop('status_object_id', None)
        self.status_record = kwargs.pop('status_record', None)

        super(UpdateStatusError, self).__init__(self.fault_string,
```

(continues on next page)

(continued from previous page)

```
                                                    *args, **kwargs)


class UpdateStatisticsError(Exception):
    fault_string = _("The statistics update had an unknown error.")
    stats_object = None
    stats_object_id = None
    stats_record = None

    def __init__(self, *args, **kwargs):
        self.fault_string = kwargs.pop('fault_string',
                                       self.fault_string)
        self.stats_object = kwargs.pop('stats_object', None)
        self.stats_object_id = kwargs.pop('stats_object_id', None)
        self.stats_record = kwargs.pop('stats_record', None)

        super(UpdateStatisticsError, self).__init__(self.fault_string,
                                                    *args, **kwargs)


class DriverAgentNotFound(Exception):
  fault_string = _("The driver-agent process was not found or not ready.")

  def __init__(self, *args, **kwargs):
      self.fault_string = kwargs.pop('fault_string', self.fault_string)
      super(DriverAgentNotFound, self).__init__(self.fault_string,
                                                *args, **kwargs)


class DriverAgentTimeout(Exception):
  fault_string = _("The driver-agent timeout.")

  def __init__(self, *args, **kwargs):
      self.fault_string = kwargs.pop('fault_string', self.fault_string)
      super(DriverAgentTimeout, self).__init__(self.fault_string,
                                               *args, **kwargs)
```

Provider Agents

Provider agents are long-running processes started by the Octavia driver-agent process at start up. They are intended to allow provider drivers a long running process that can handle periodic jobs for the provider driver or receive events from another provider agent. Provider agents are optional and not required for a successful Octavia provider driver.

Provider Agents have access to the same *Stable Provider Driver Interface* as the provider driver. A provider agent must not access any other Octavia code.

> **Warning:** The methods listed in the *Driver Support Library* section are the only Octavia callable methods for provider agents. All other interfaces are not considered stable or safe for provider agents to access. See *Stable Provider Driver Interface* for a list of acceptable APIs for provider agents use.

Declaring Your Provider Agent

The Octavia driver-agent will use stevedore to load enabled provider agents at start up. Provider agents are enabled in the Octavia configuration file. Provider agents that are installed, but not enabled, will not be loaded. An example configuration file entry for a provider agent is:

```
[driver_agent]
enabled_provider_agents = amphora_agent, noop_agent
```

The provider agent name must match the provider agent name declared in your python setup tools entry point. For example:

```
octavia.driver_agent.provider_agents =
    amphora_agent = octavia.api.drivers.amphora_driver.
↪agent:AmphoraProviderAgent
    noop_agent = octavia.api.drivers.noop_driver.agent:noop_provider_agent
```

Provider Agent Method Invocation

On start up of the Octavia driver-agent, the method defined in the entry point will be launched in its own multiprocessing Process.

Your provider agent method will be passed a multiprocessing Event that will be used to signal that the provider agent should shutdown. When this event is "set", the provider agent should gracefully shutdown. If the provider agent fails to exit within the Octavia configuration file setting "provider_agent_shutdown_timeout" period, the driver-agent will forcefully shutdown the provider agent with a SIGKILL signal.

Example Provider Agent Method

If, for example, you declared a provider agent as "my_agent":

```
octavia.driver_agent.provider_agents =
    my_agent = example_inc.drivers.my_driver.agent:my_provider_agent
```

The signature of your "my_provider_agent" method would be:

```
def my_provider_agent(exit_event):
```

Documenting the Driver

Octavia provides two documents to let operators and users know about available drivers and their features.

Available Provider Drivers

The *Available Provider Drivers* document provides administrators with a guide to the available Octavia provider drivers. Since provider drivers are not included in the Octavia source repositories, this guide is an important tool for administrators to find your provider driver.

You can submit information for your provider driver by submitting a patch to the Octavia documentation following the normal OpenStack process.

See the OpenStack Contributor Guide for more information on submitting a patch to OpenStack.

Octavia Provider Feature Matrix

The Octavia documentation includes a *Octavia Provider Feature Matrix* that informs users on which Octavia features are supported by each provider driver.

The feature matrices are built using the Oslo sphinx-feature-classification library. This allows a simple INI file format for describing the capabilities of an Octavia provider driver.

Each driver should define a [driver.<driver name>] section and then add a line to each feature specifying the level of support the provider driver provides for the feature.

For example, the Amphora driver support for "admin_state_up" would add the following to the feature-matrix-lb.ini file.

```
[driver.amphora]
title=Amphora Provider
link=https://docs.openstack.org/api-ref/load-balancer/v2/index.html

[operation.admin_state_up]
...
driver.amphora=complete
```

Valid driver feature support statuses are:

`complete` Fully implemented, expected to work at all times.

`partial` Implemented, but with caveats about when it will work.

`missing` Not implemented at all.

You can also optionally provide additional, provider driver specific, notes for users by defining a "driver-notes.<driver name>".

```
[operation.admin_state_up]
...
driver.amphora=complete
driver-notes.amphora=The Amphora driver fully supports admin_state_up.
```

Driver notes are highly recommended when a provider driver declares a `partial` status.

4.2.2 Debugging Octavia code

This document describes how to setup and debug Octavia code using your favorite IDE (e.g. PyCharm, Visual Studio Code).

Prerequisites

- Octavia installed.
- IDE installed and Octavia added as project.

Setup

Ensure your OpenStack and IDE environments have the PyDev or ptvsd library installed.

If you're using PyCharm, you can find it in */path/to/pycharm/debug-eggs/pycharm-debug.egg* (Python 2) and */path/to/pycharm/debug-eggs/pycharm-debug-py3k.egg* (Python 3). Copy that file into your OpenStack host and install the library in your Python path:

```
$ sudo easy_install pycharm-debug.egg
```

If using Visual Studio Code, simply install ptvsd in both environments:

```
$ pip install ptvsd
```

Create a remote debugging configuration in your IDE. In PyCharm, go to *Run -> Edit Configurations -> Python Remote Debug*. The local host name refers to the local machine you're running your IDE from and it must be one reachable by your OpenStack environment. The port can be any available port (e.g. 5678). If the code on the OpenStack and PyCharm hosts is on different paths (likely), define a path mapping in the remote debug configuration.

Invoke the debug configuration (*Run -> Debug... -> (config name)*). PyCharm will begin listening on the specified host and port.

Export *DEBUGGER_TYPE*, *DEBUGGER_HOST* and *DEBUGGER_PORT* (host and port of the system running the IDE, respectively), and start the Octavia service you want to debug. It is recommended to run only one uWSGI process/controller worker. For example, to debug the Octavia Worker service:

```
$ export DEBUGGER_TYPE=pydev
$ export DEBUGGER_HOST=192.168.121.1
$ export DEBUGGER_PORT=5678
$ /usr/bin/octavia-worker --config-file /etc/octavia/octavia.conf
```

Another example is debugging the Octavia API service with the ptvsd debugger:

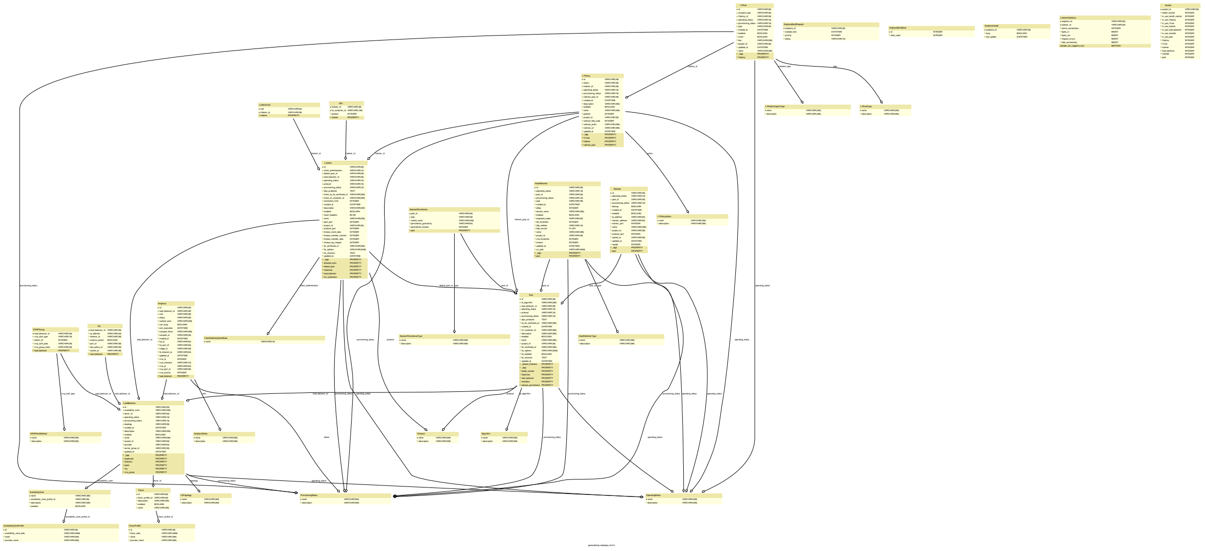
```
$ export DEBUGGER_TYPE=ptvsd
$ export DEBUGGER_HOST=192.168.121.1
$ export DEBUGGER_PORT=5678
$ /usr/bin/uwsgi --ini /etc/octavia/octavia-uwsgi.ini -p 1
```

The service will connect to your IDE, at which point remote debugging is active. Resume the program on the debugger to continue with the initialization of the service. At this point, the service should be operational and you can start debugging.

4.2.3 Octavia Entity Relationship Diagram

Below is the current Octavia database data model.

- Solid stars are primary key columns.
- Hollow stars are foreign key columns.
- Items labeled as "PROPERTY" are data model relationships and are not present in the database.



4.2.4 Octavia Controller Flows

Octavia uses OpenStack TaskFlow to orchestrate the actions the Octavia controller needs to take while managing load balancers.

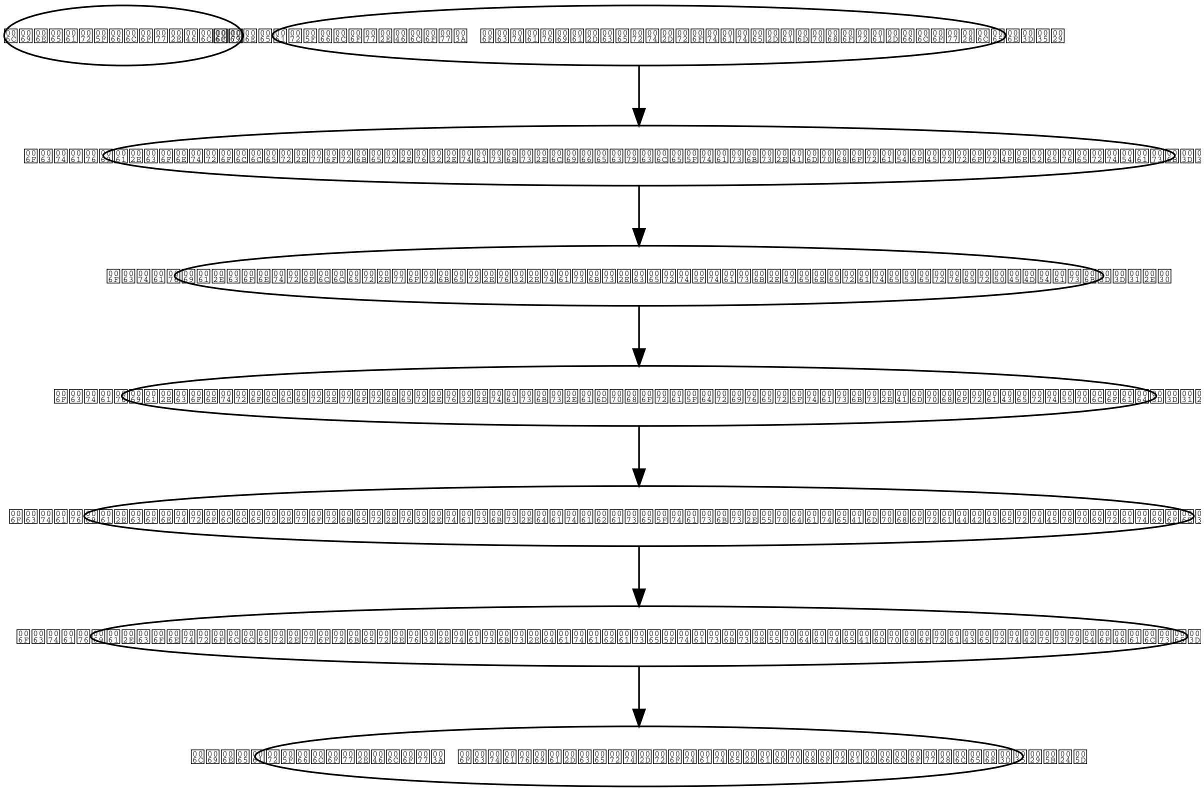
This document is meant as a reference for the key flows used in the Octavia controller.

Amphora Flows

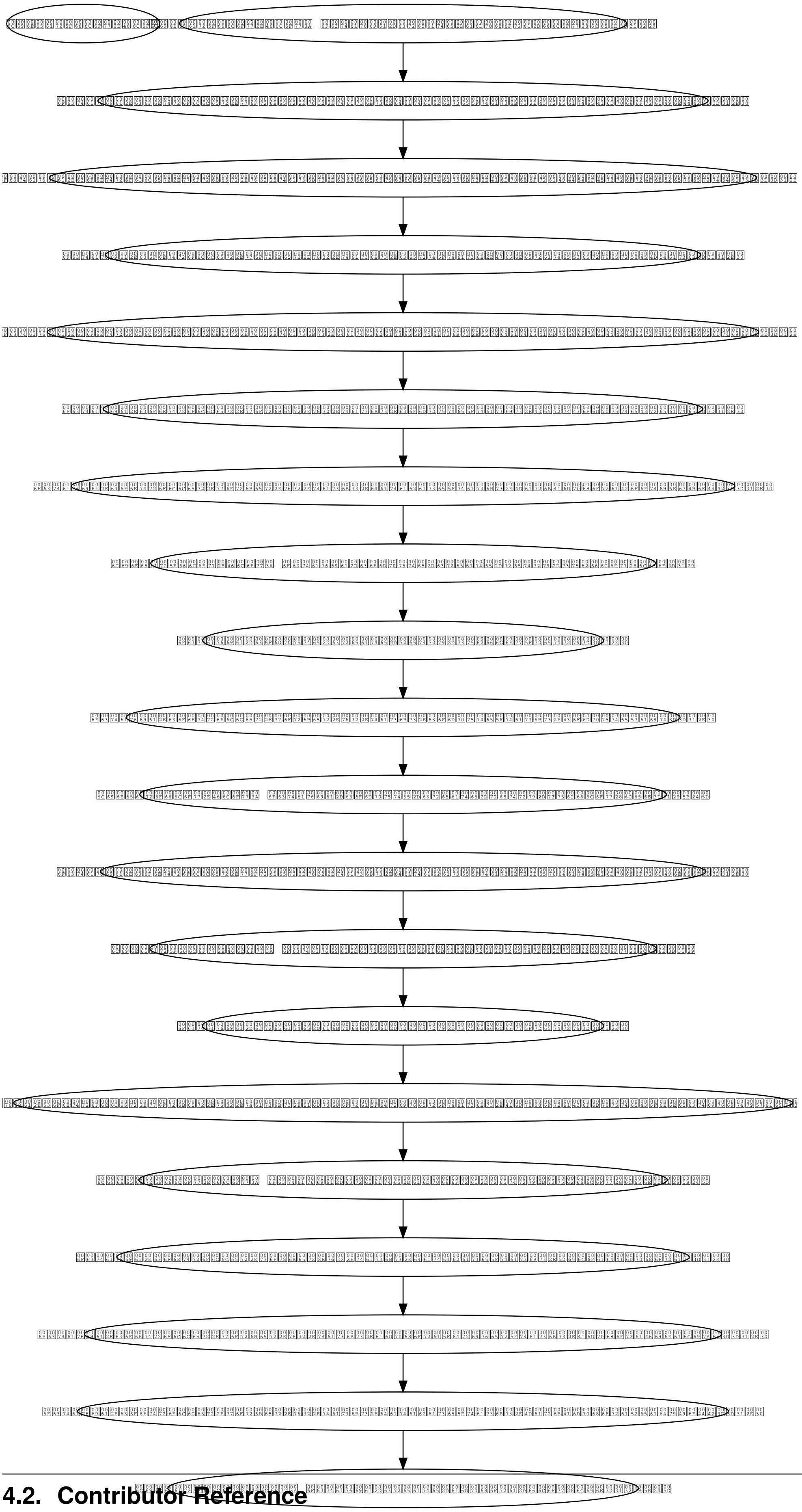
Contents

- *Amphora Flows*
  - *cert_rotate_amphora_flow*
  - *get_create_amphora_flow*
  - *get_failover_amphora_flow*

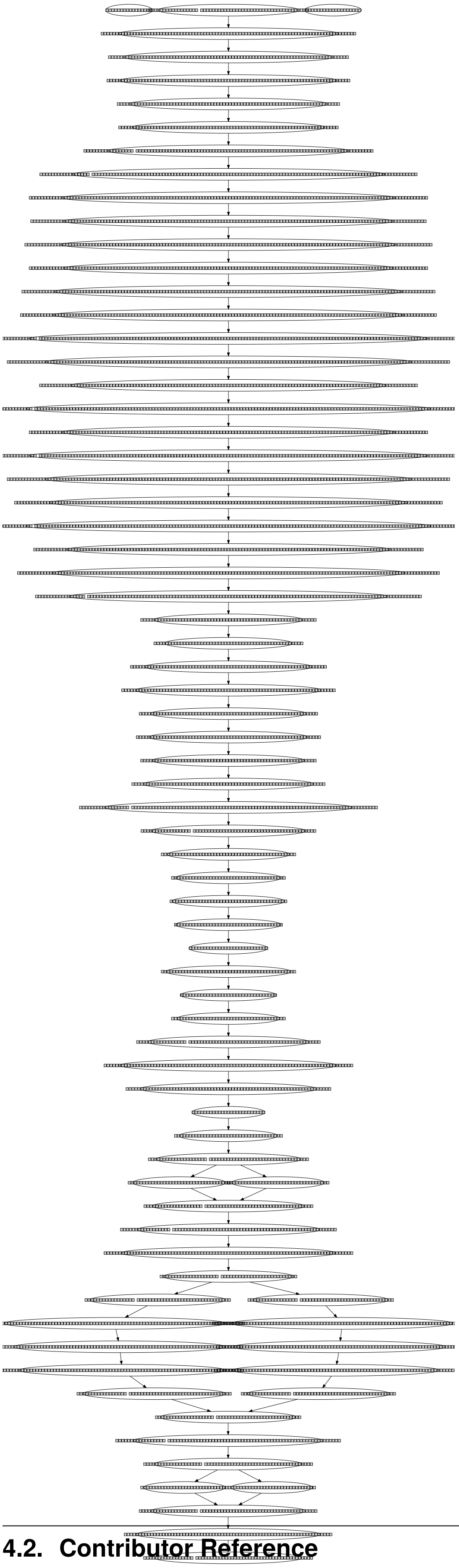
cert_rotate_amphora_flow



get_create_amphora_flow



get_failover_amphora_flow

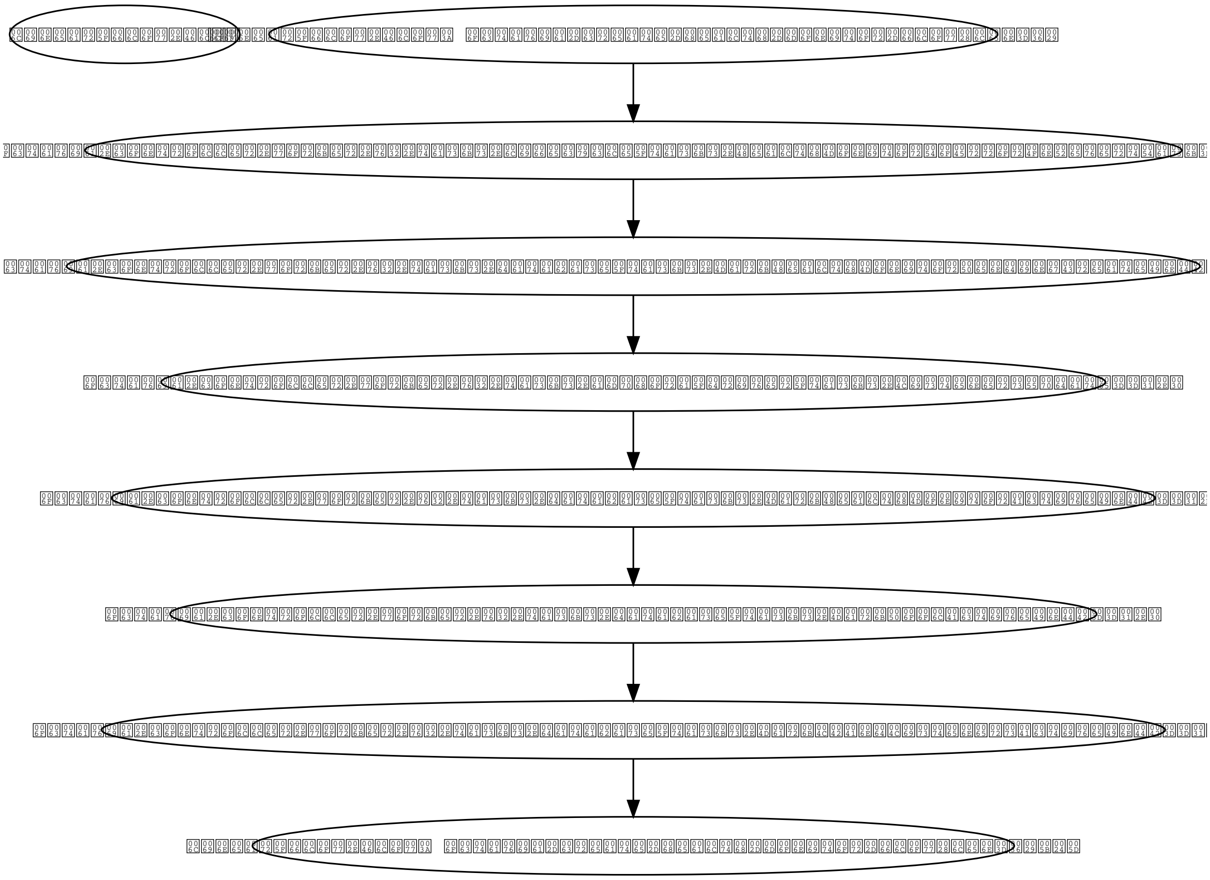


Health Monitor Flows

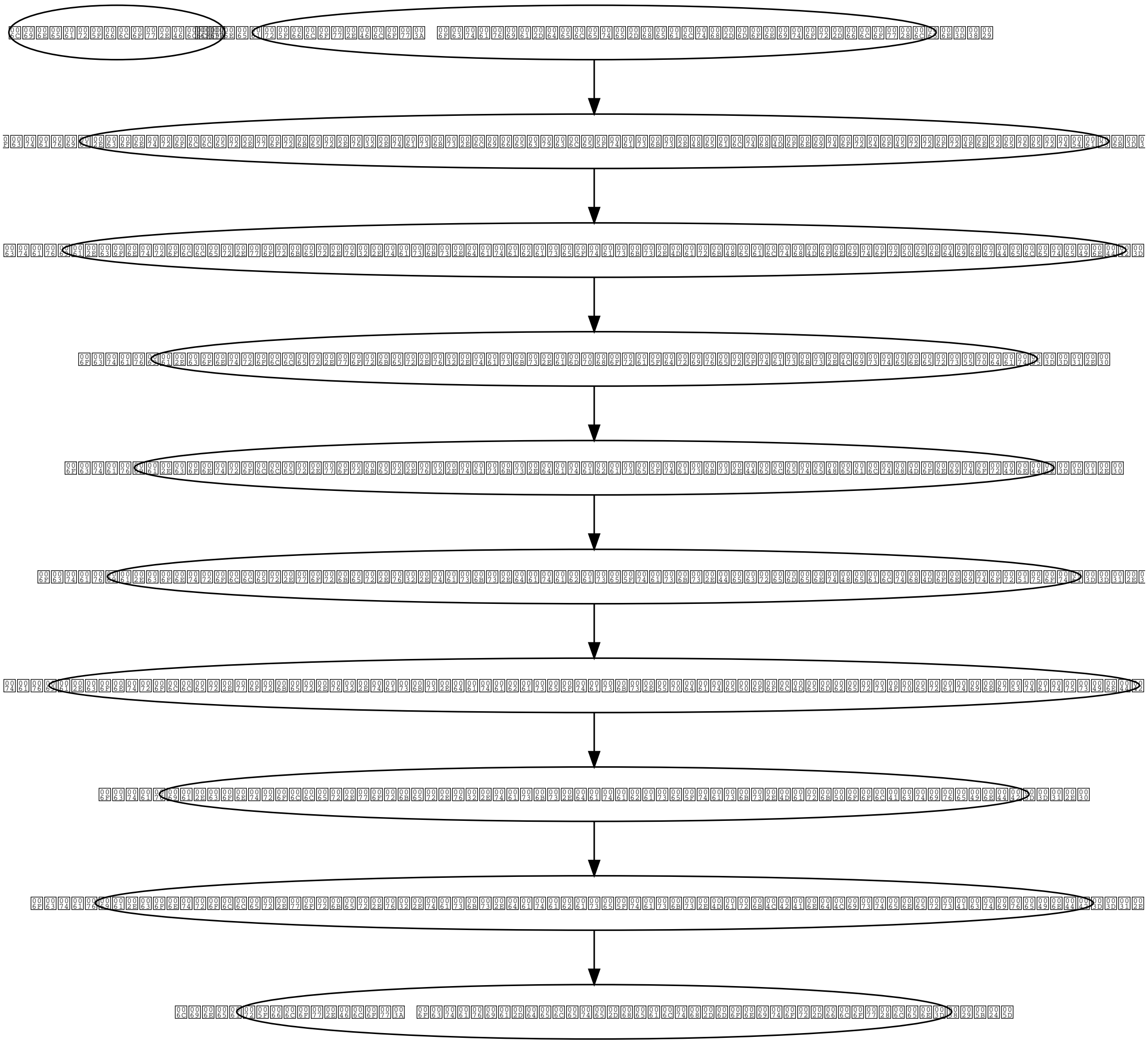
Contents

- *Health Monitor Flows*
  - *get_create_health_monitor_flow*
  - *get_delete_health_monitor_flow*
  - *get_update_health_monitor_flow*

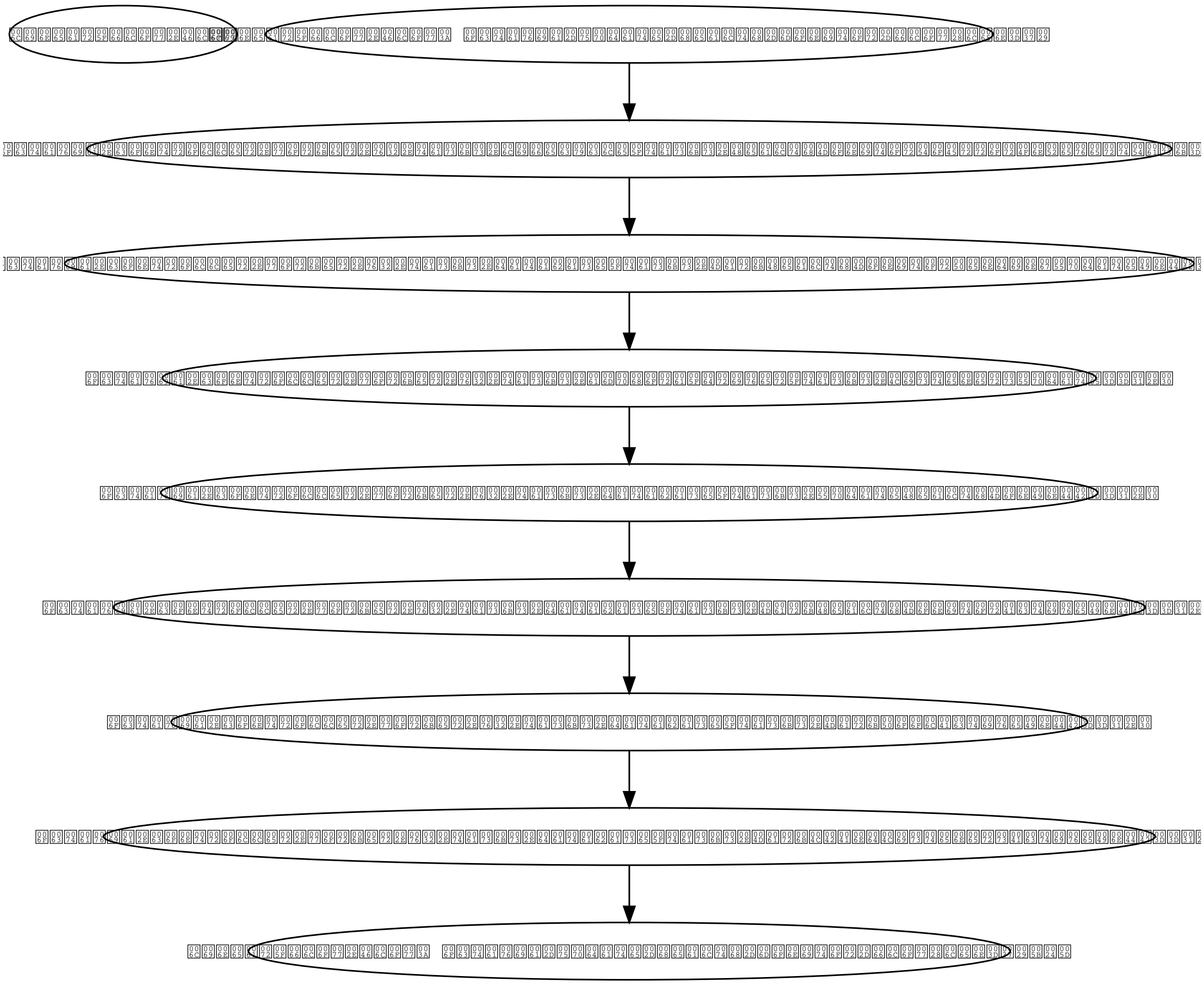
get_create_health_monitor_flow



get_delete_health_monitor_flow



get_update_health_monitor_flow

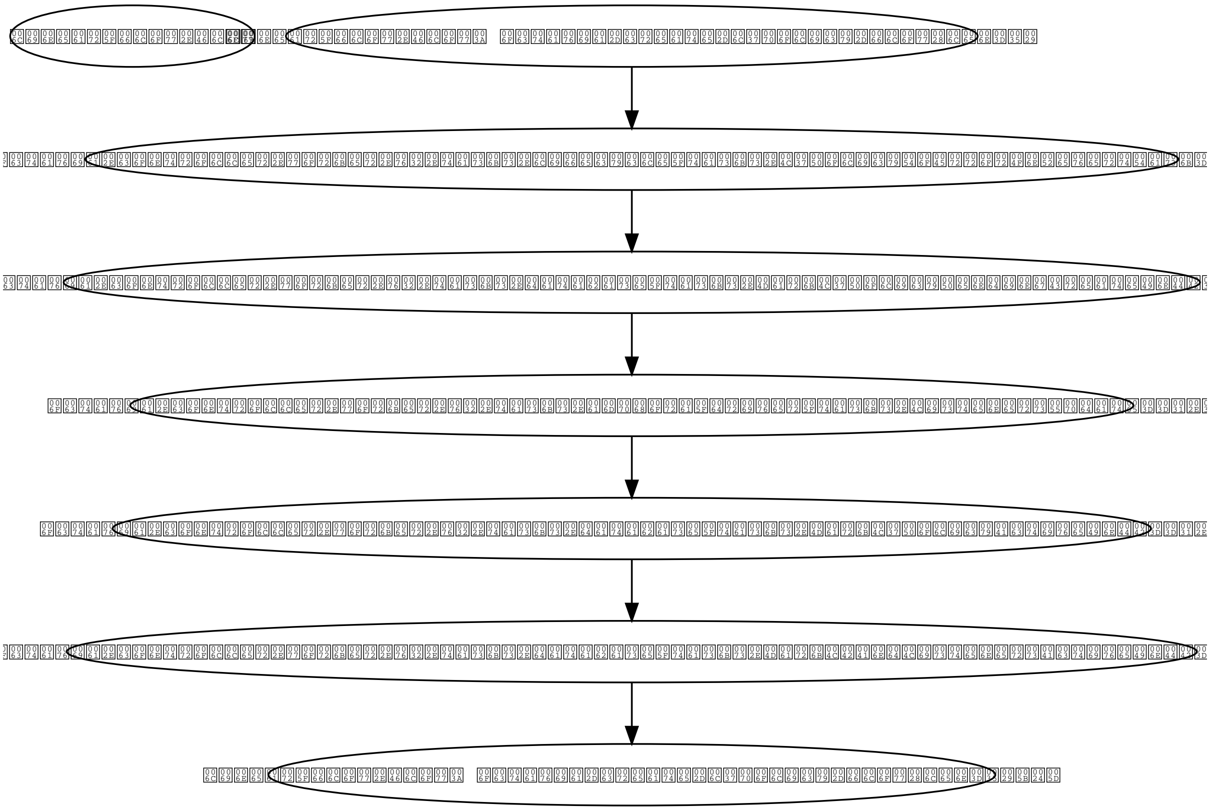


Layer 7 Policy Flows

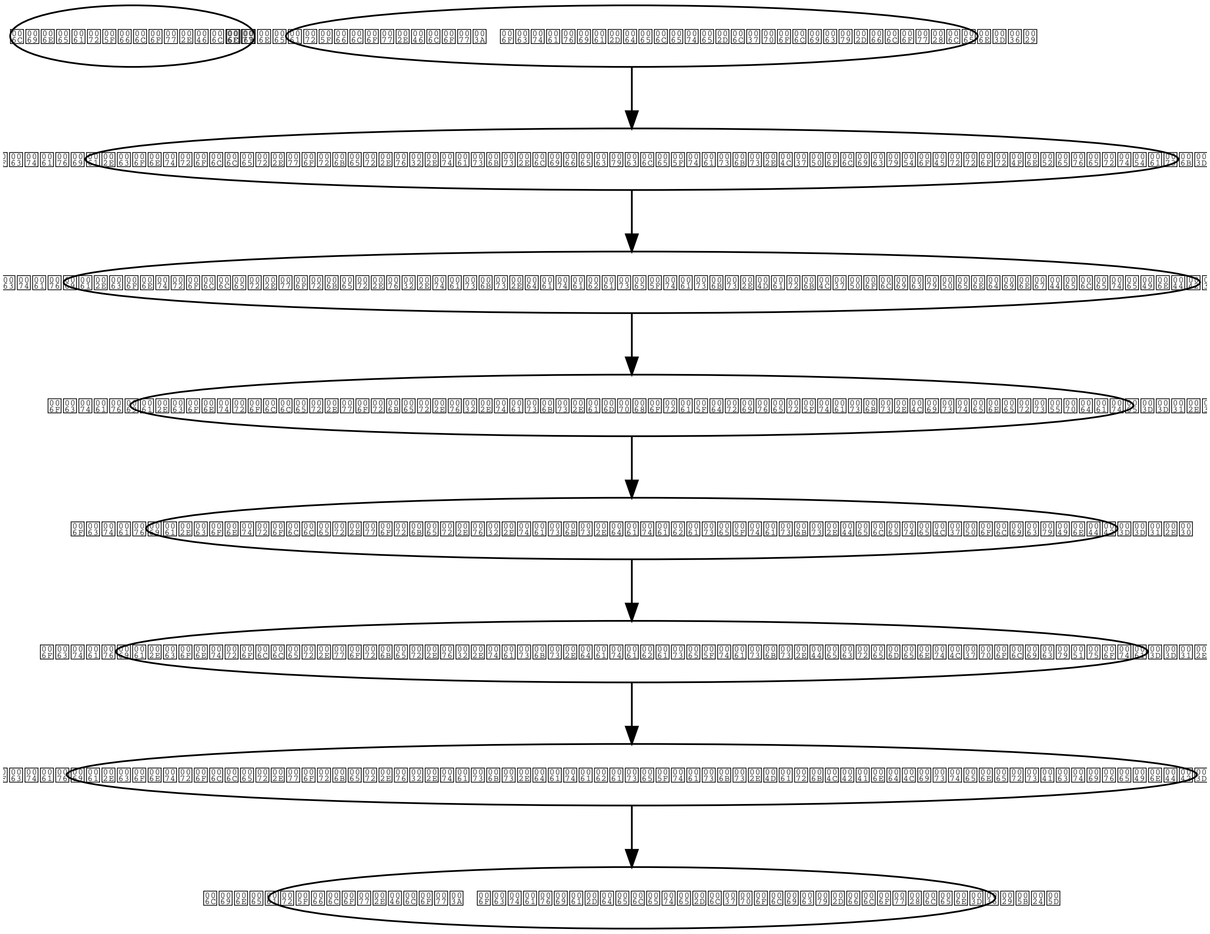
Contents

- *Layer 7 Policy Flows*
  - *get_create_l7policy_flow*
  - *get_delete_l7policy_flow*
  - *get_update_l7policy_flow*

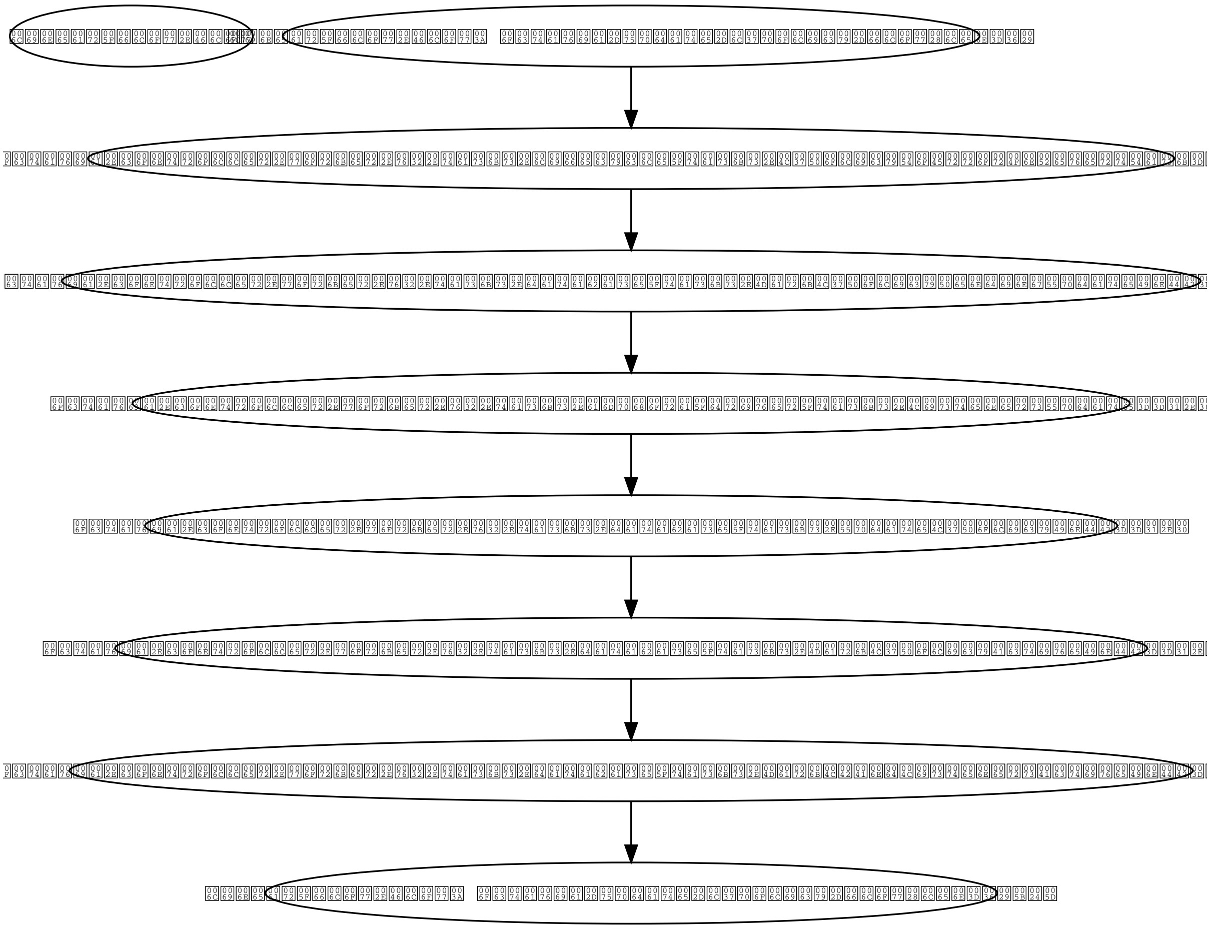
get_create_l7policy_flow



get_delete_l7policy_flow



get_update_l7policy_flow

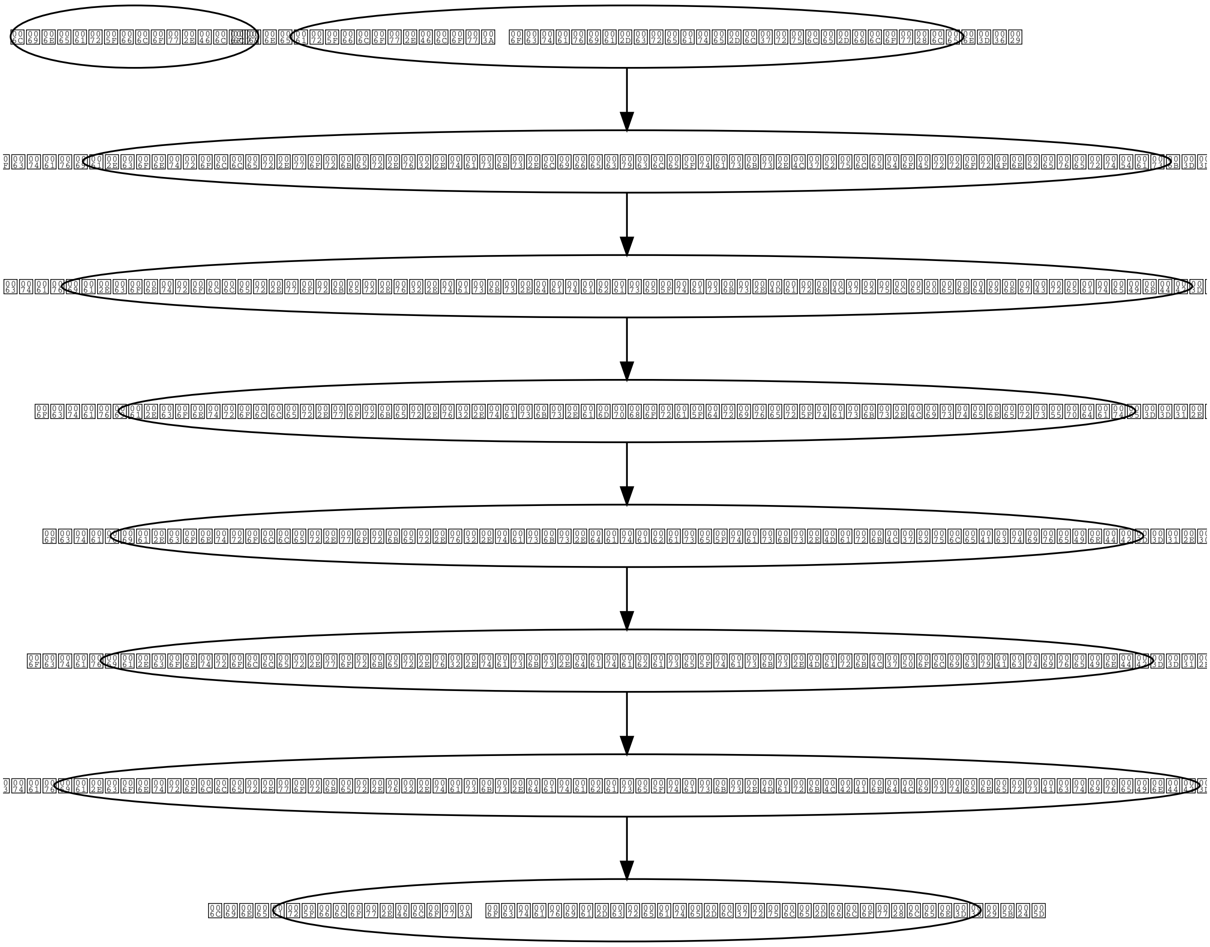


Layer 7 Rule Flows

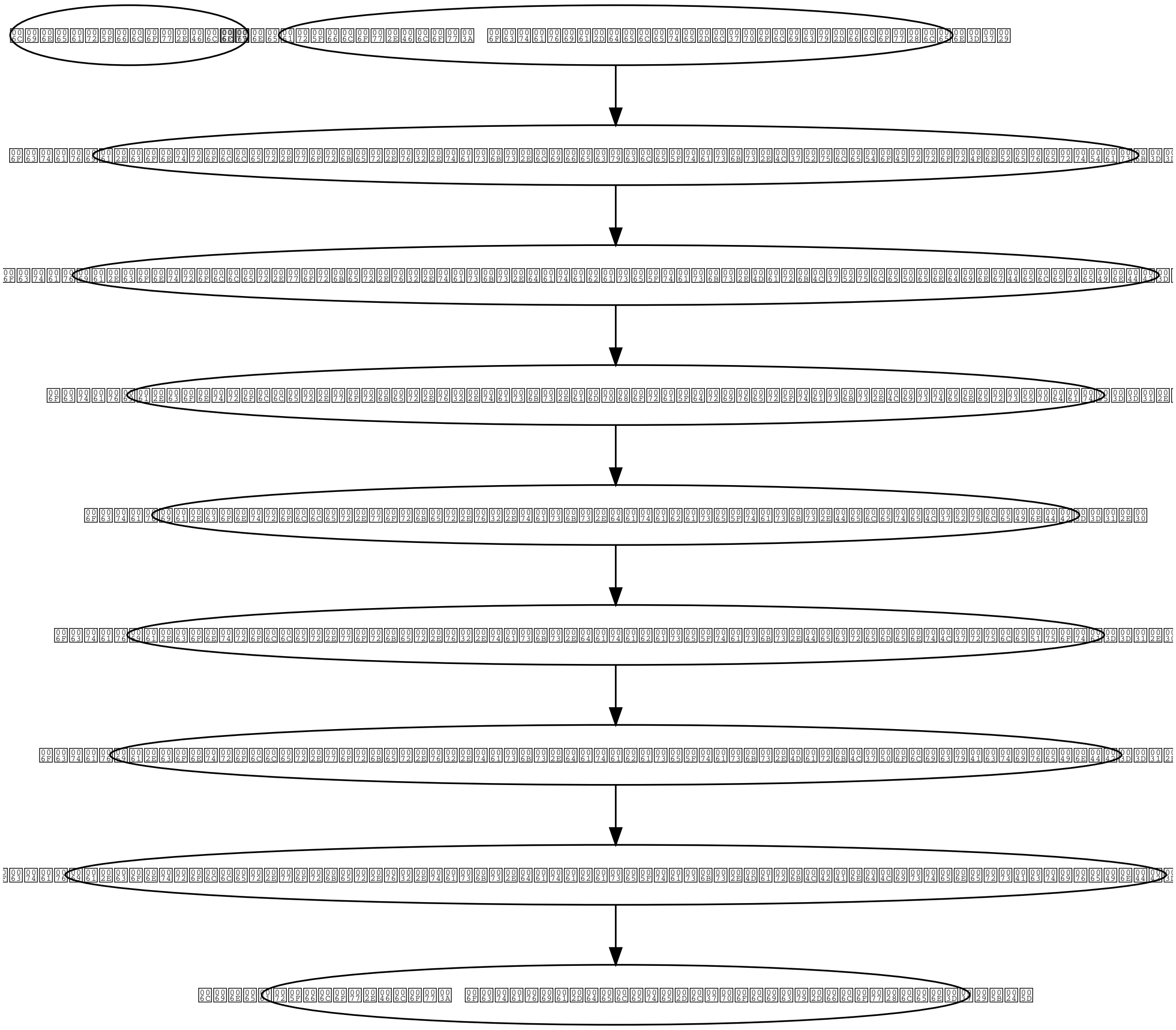
Contents

- *Layer 7 Rule Flows*
  - *get_create_l7rule_flow*
  - *get_delete_l7rule_flow*
  - *get_update_l7rule_flow*

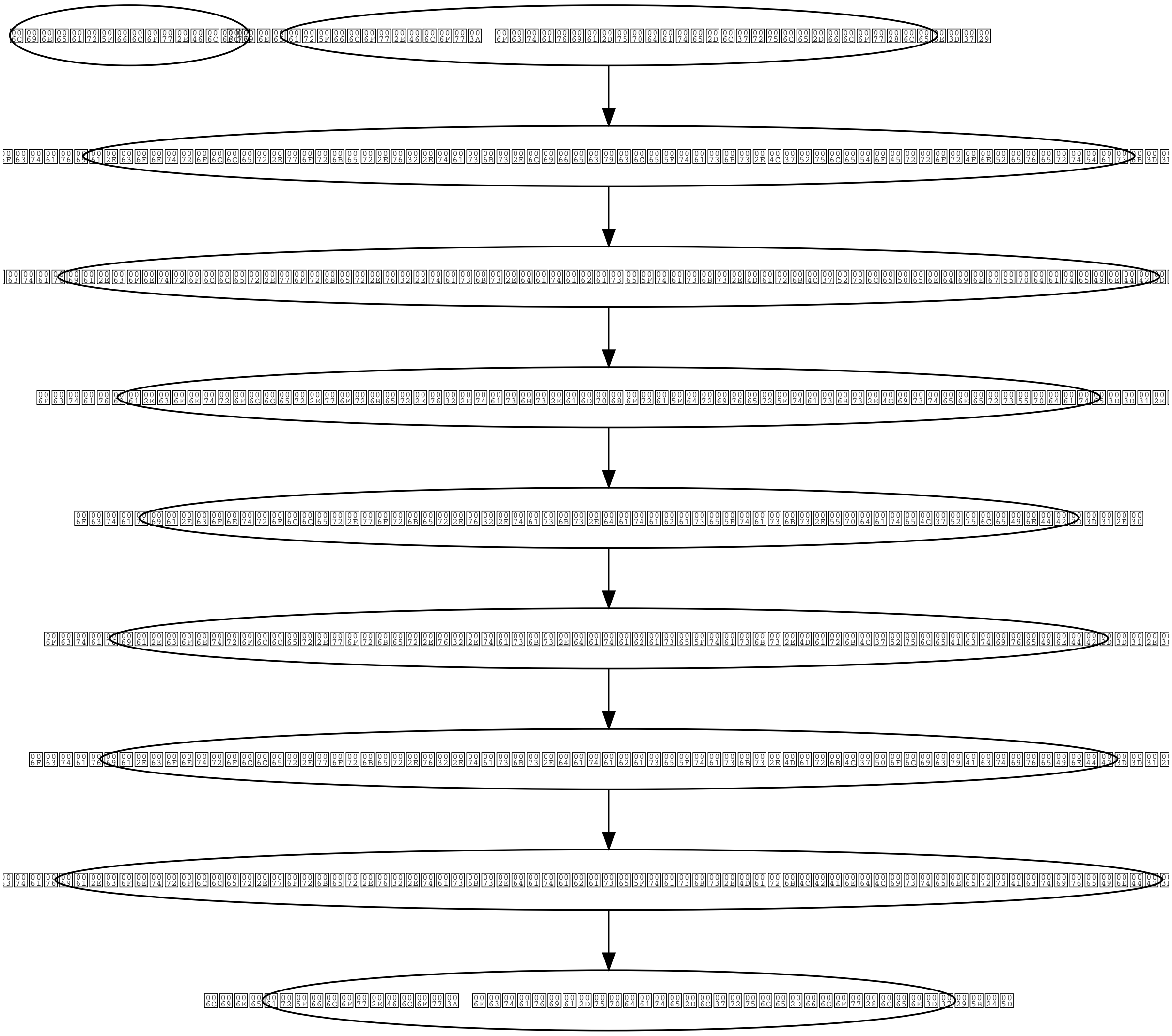
get_create_l7rule_flow



get_delete_l7rule_flow



get_update_l7rule_flow

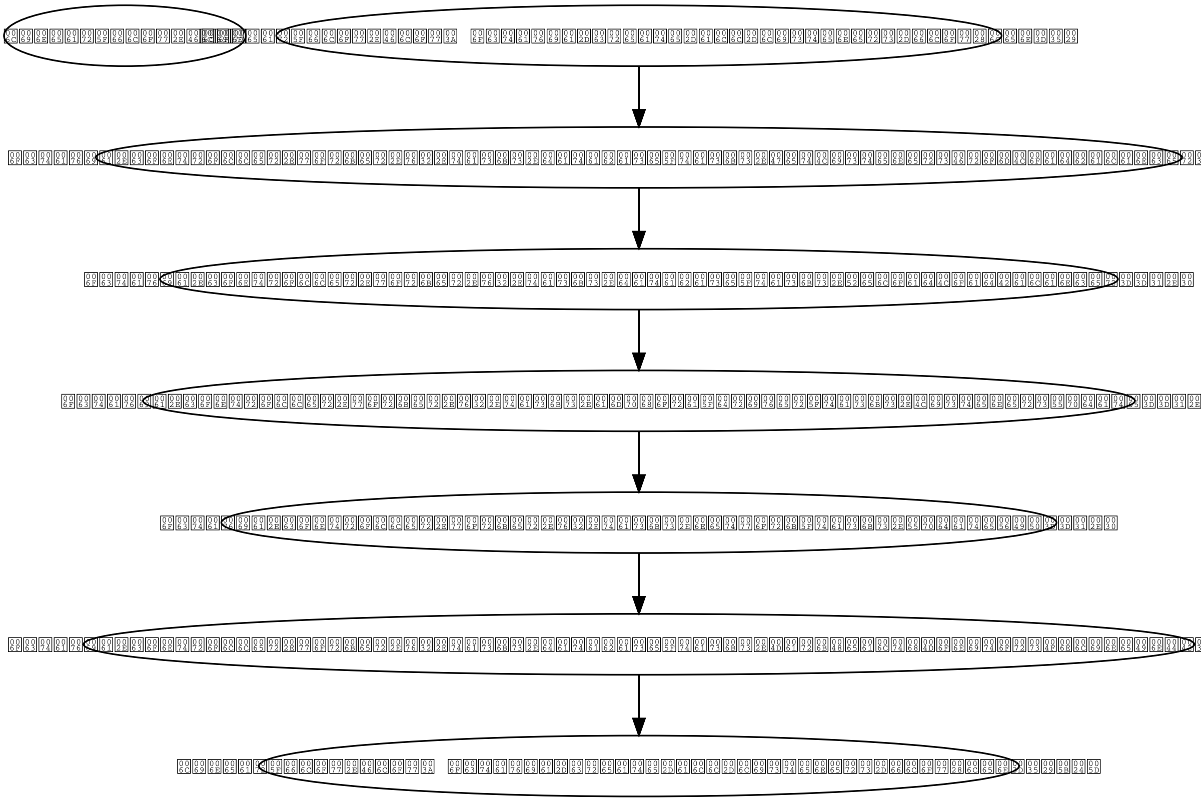


Listener Flows

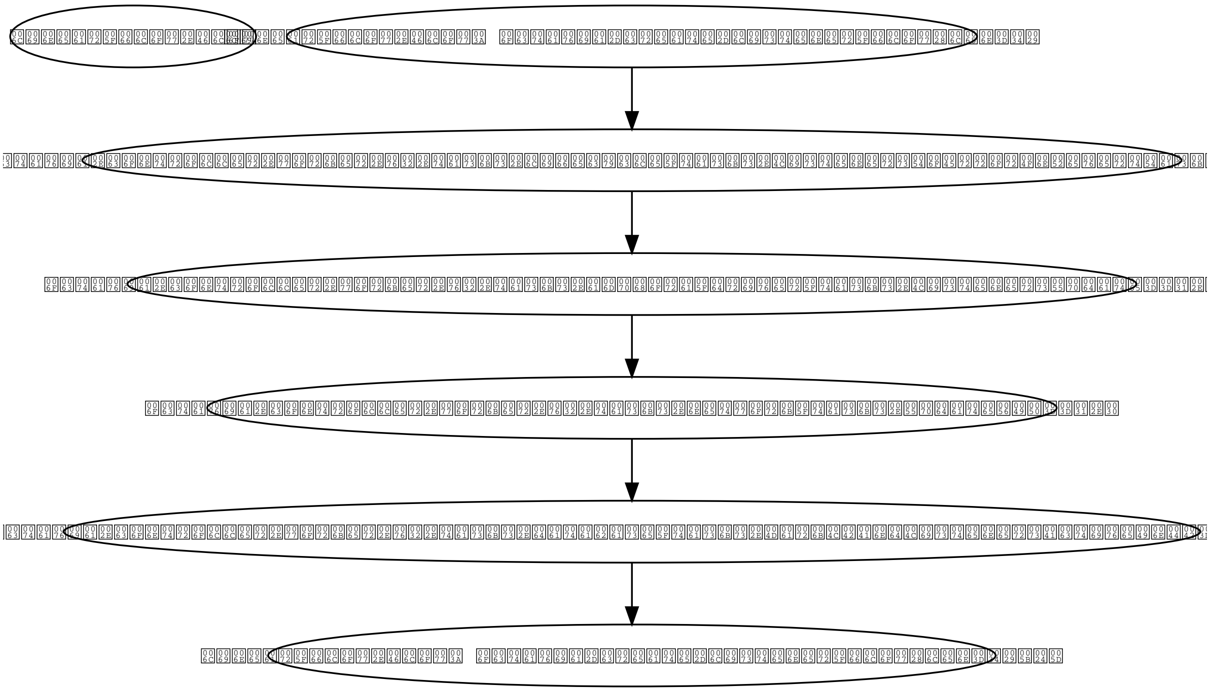
Contents

- *Listener Flows*
  - *get_create_all_listeners_flow*
  - *get_create_listener_flow*
  - *get_delete_listener_flow*
  - *get_update_listener_flow*

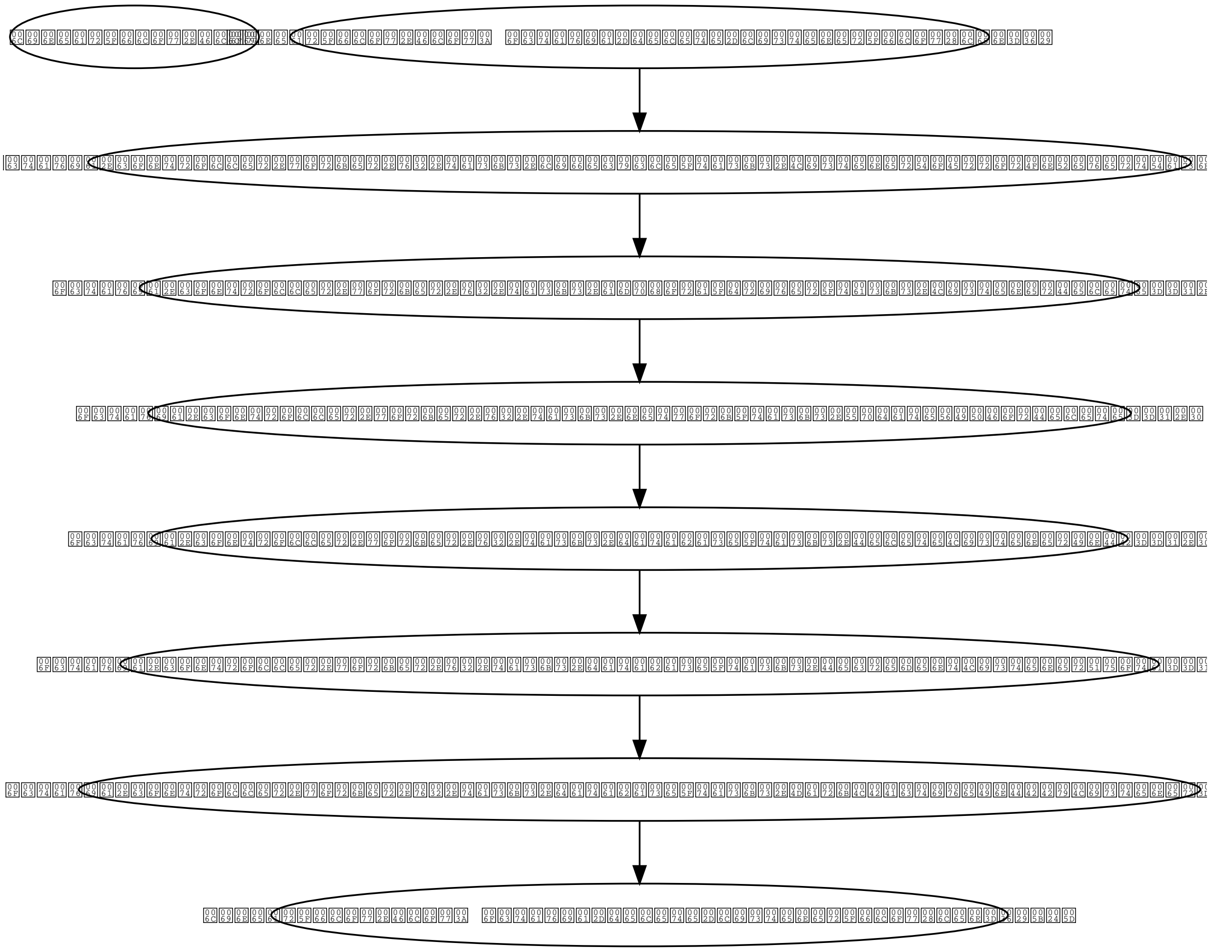
get_create_all_listeners_flow



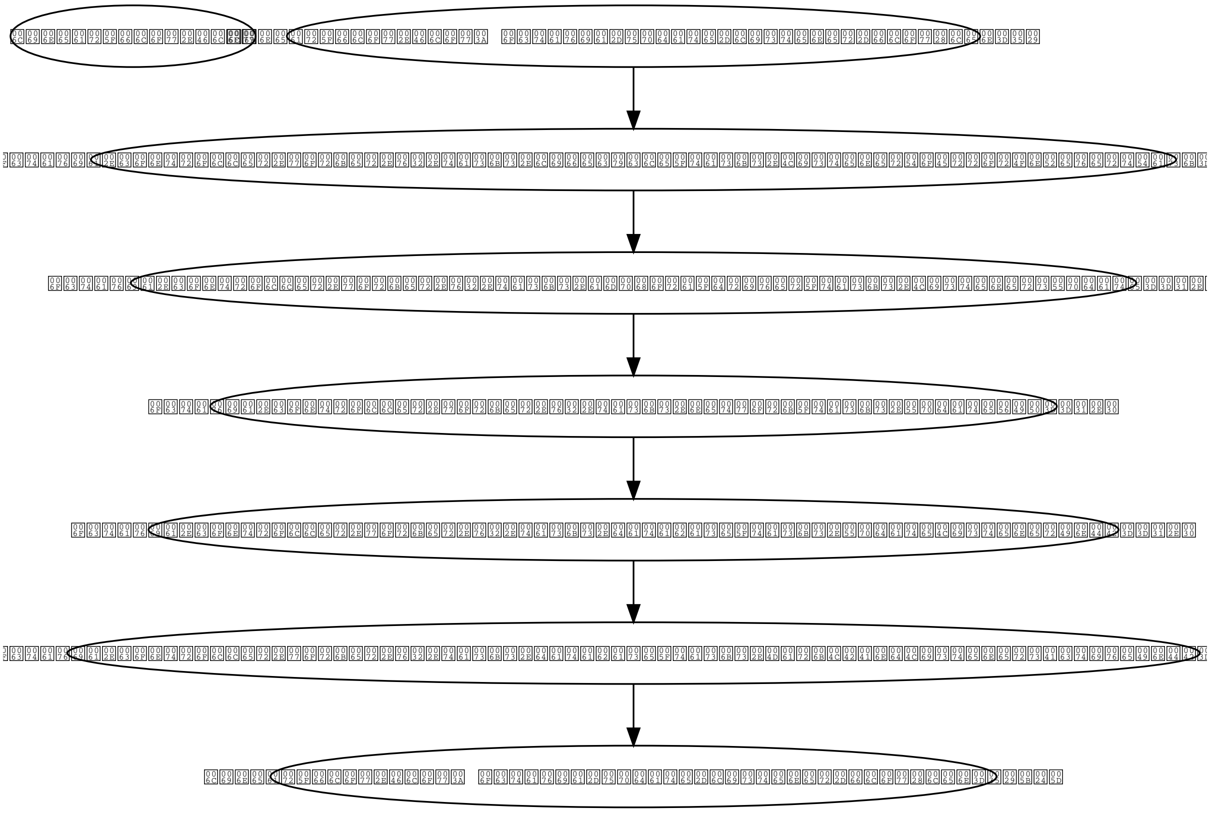
get_create_listener_flow



get_delete_listener_flow



get_update_listener_flow

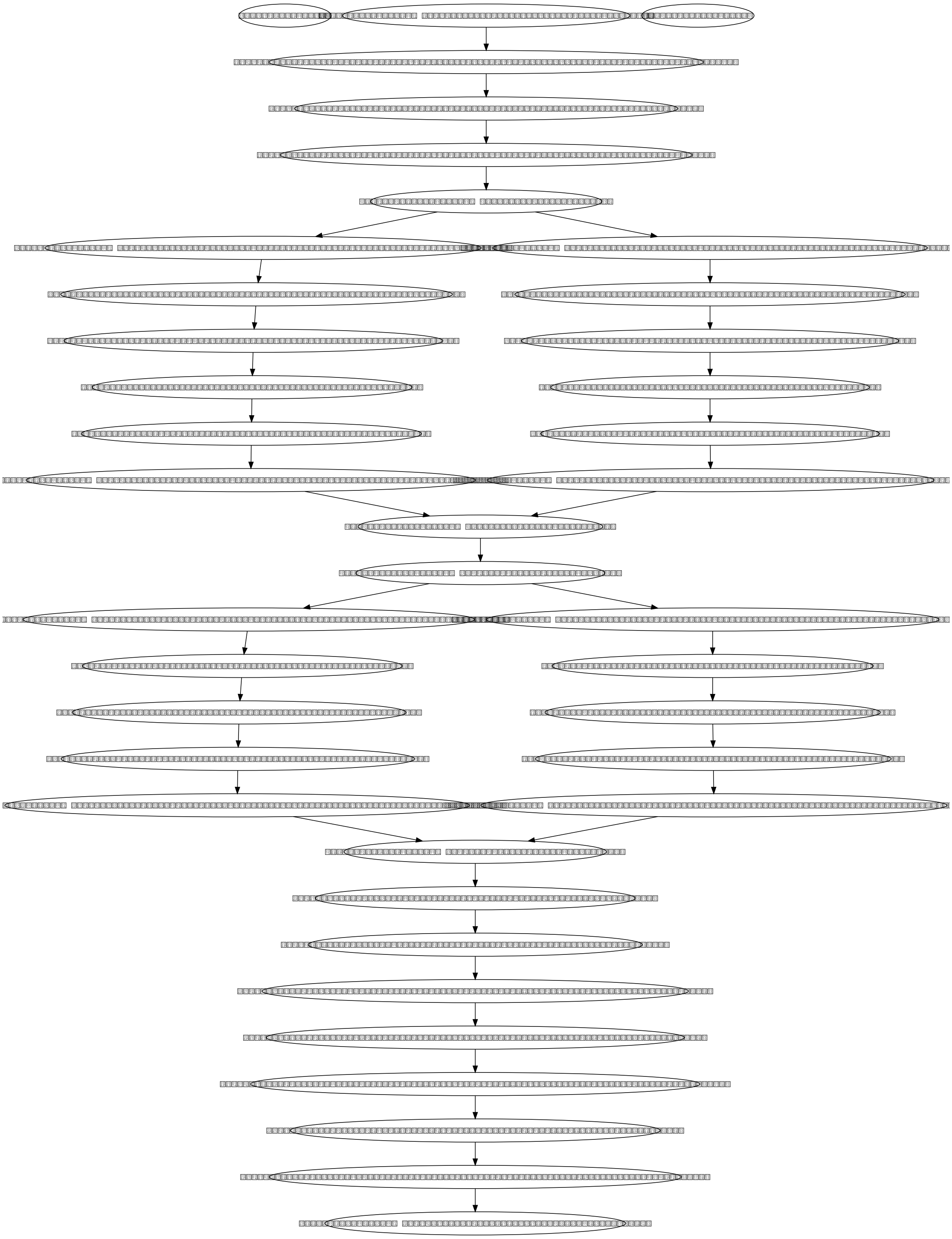


Load Balancer Flows

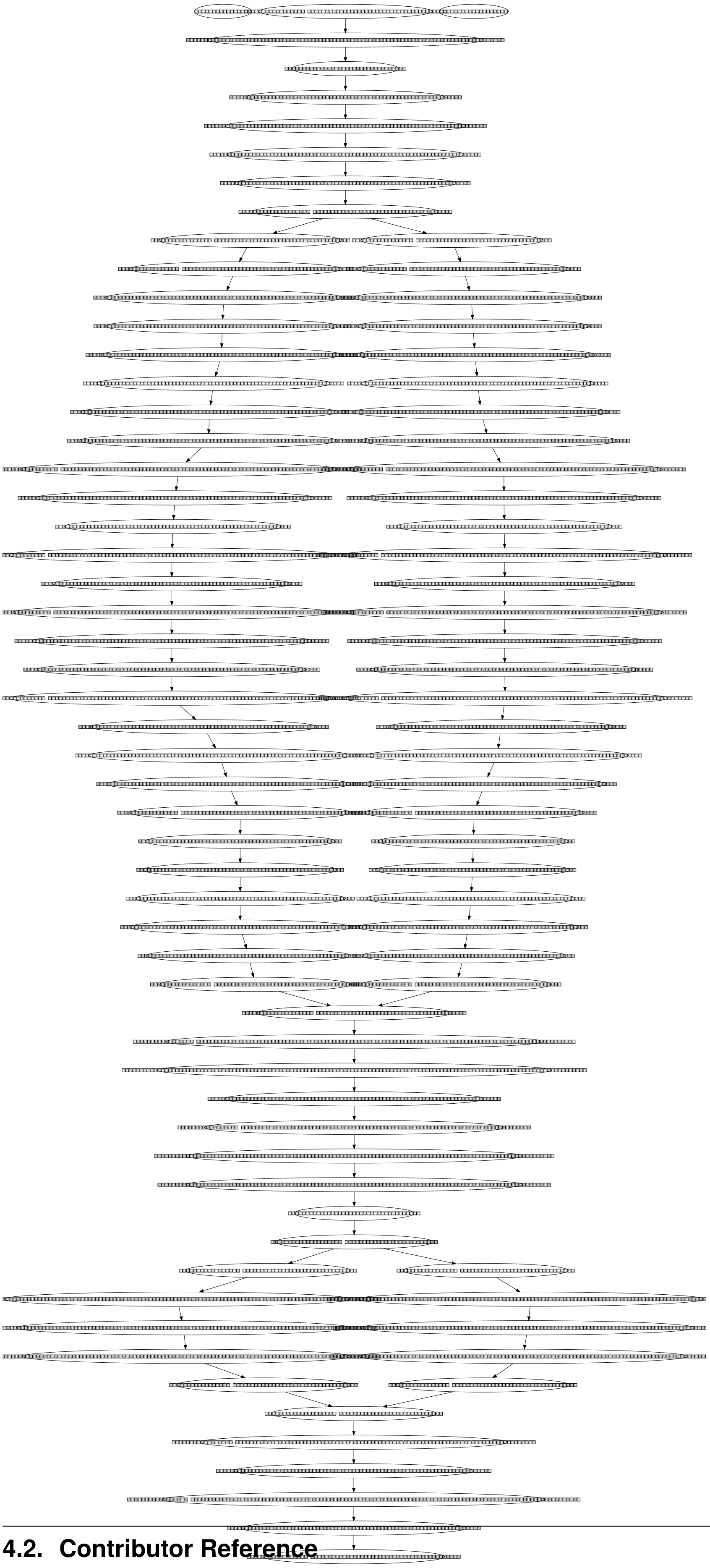
Contents

- *Load Balancer Flows*
  - *get_cascade_delete_load_balancer_flow*
  - *get_create_load_balancer_flow*
  - *get_delete_load_balancer_flow*
  - *get_failover_LB_flow*
  - *get_update_load_balancer_flow*

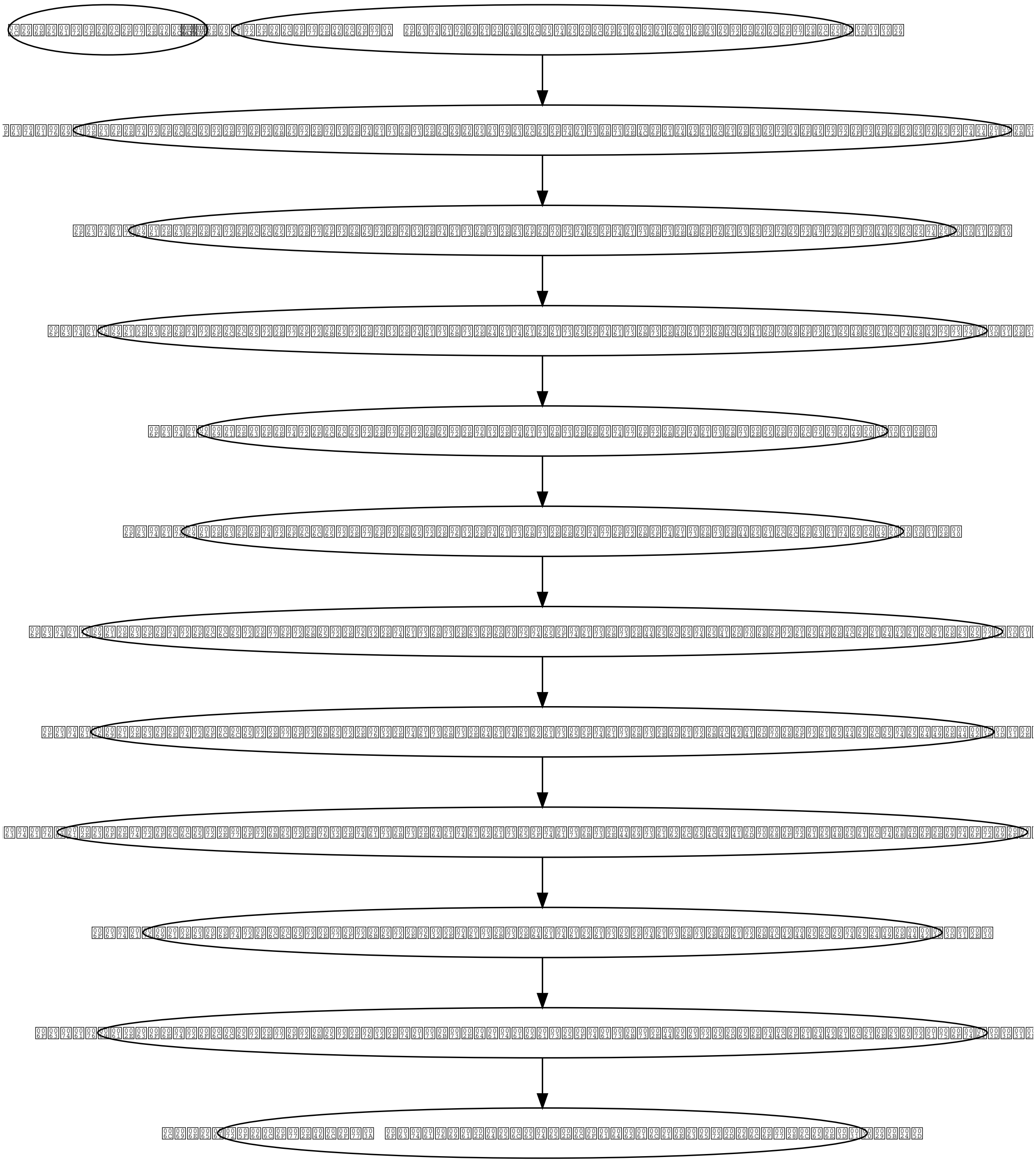
get_cascade_delete_load_balancer_flow



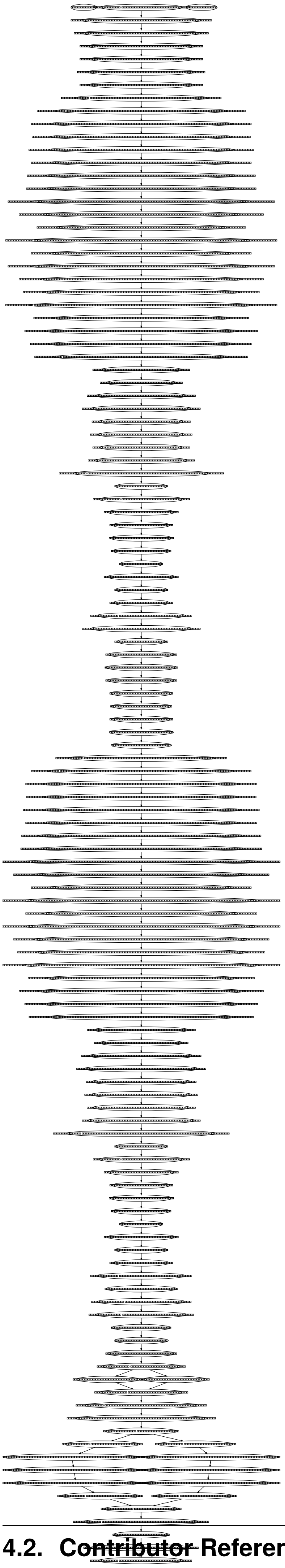
get_create_load_balancer_flow



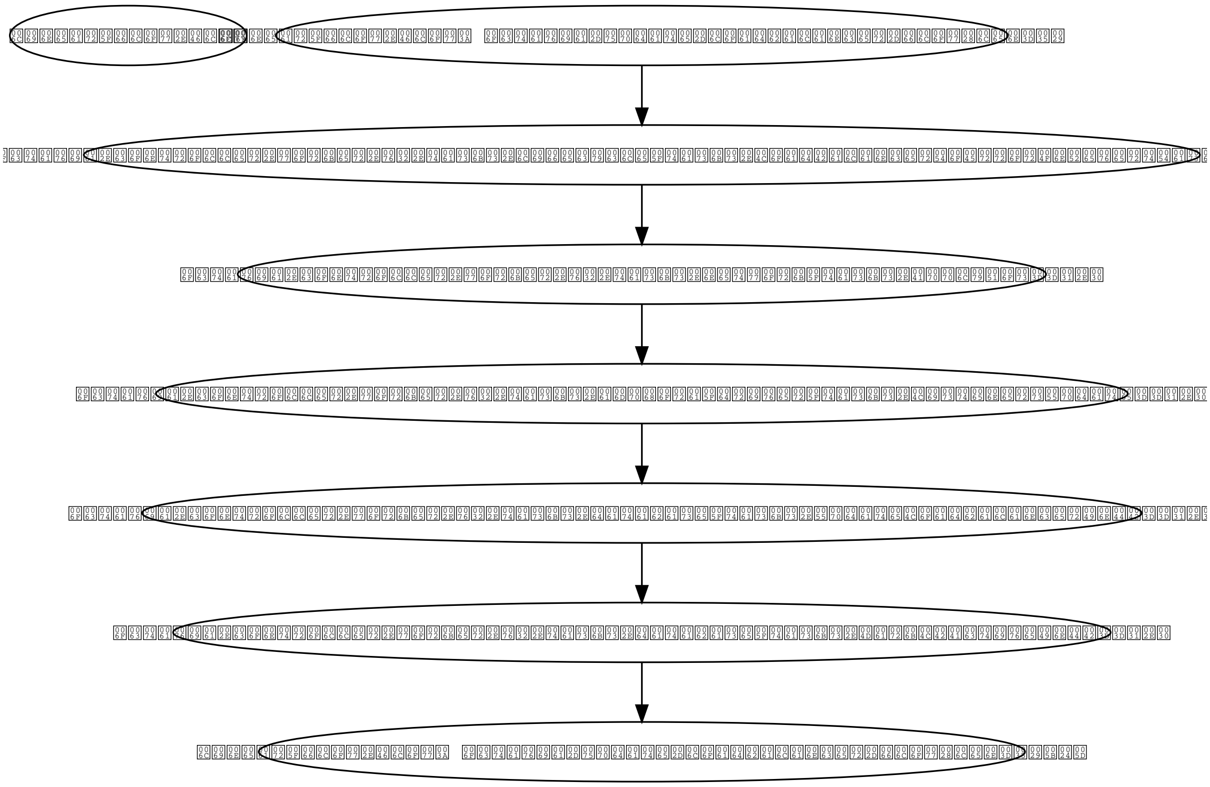
get_delete_load_balancer_flow



get_failover_LB_flow



get_update_load_balancer_flow

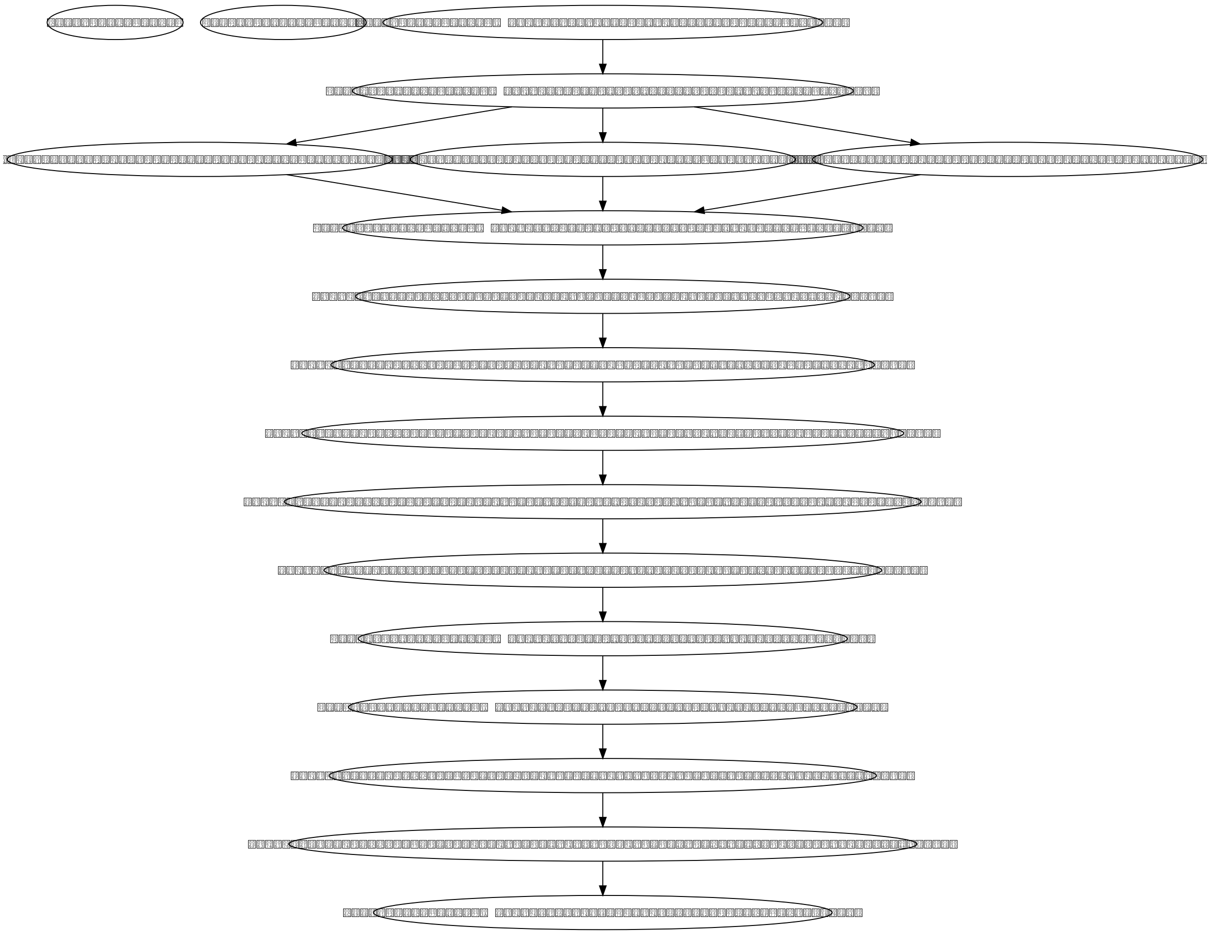


Member Flows

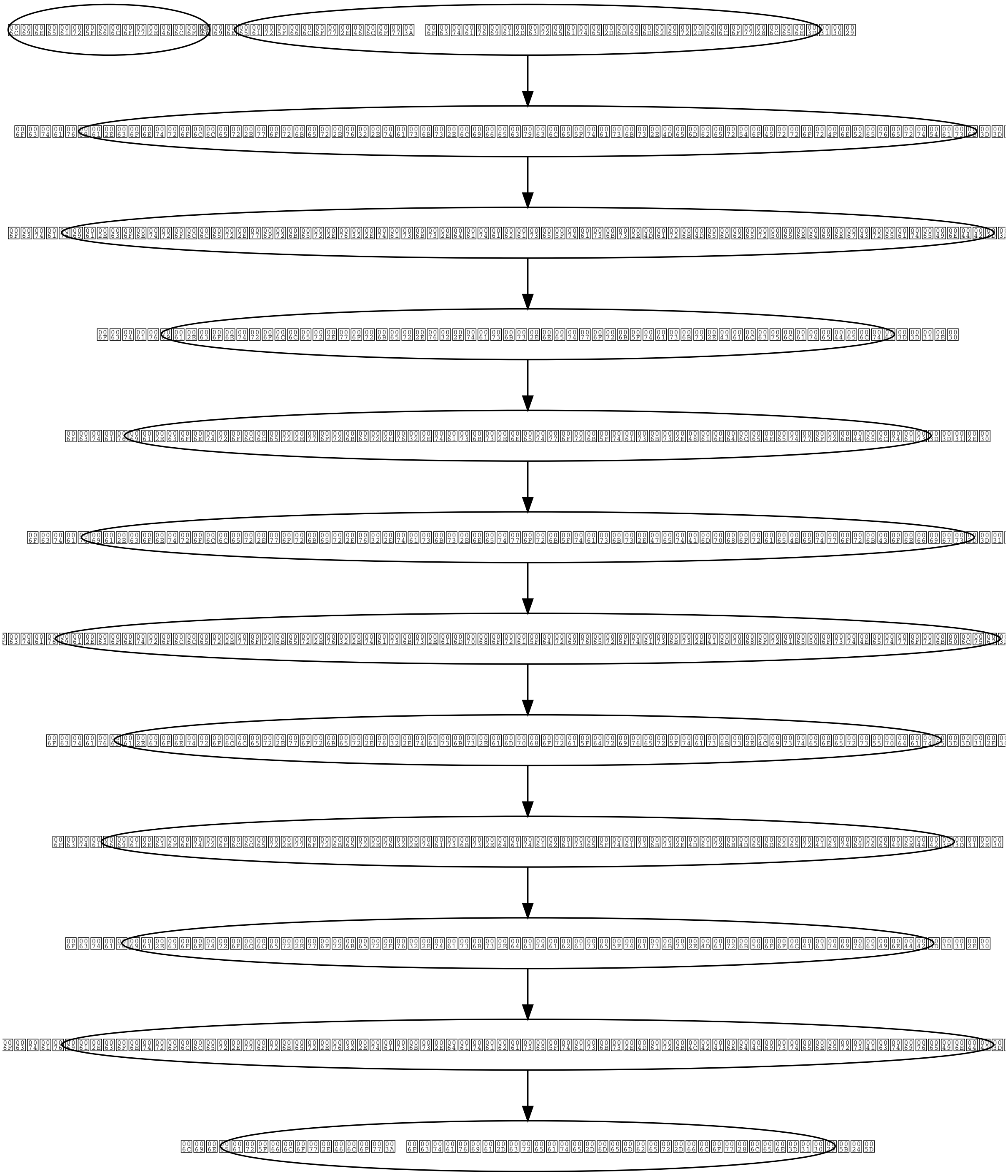
Contents

- *Member Flows*
  - *get_batch_update_members_flow*
  - *get_create_member_flow*
  - *get_delete_member_flow*
  - *get_update_member_flow*

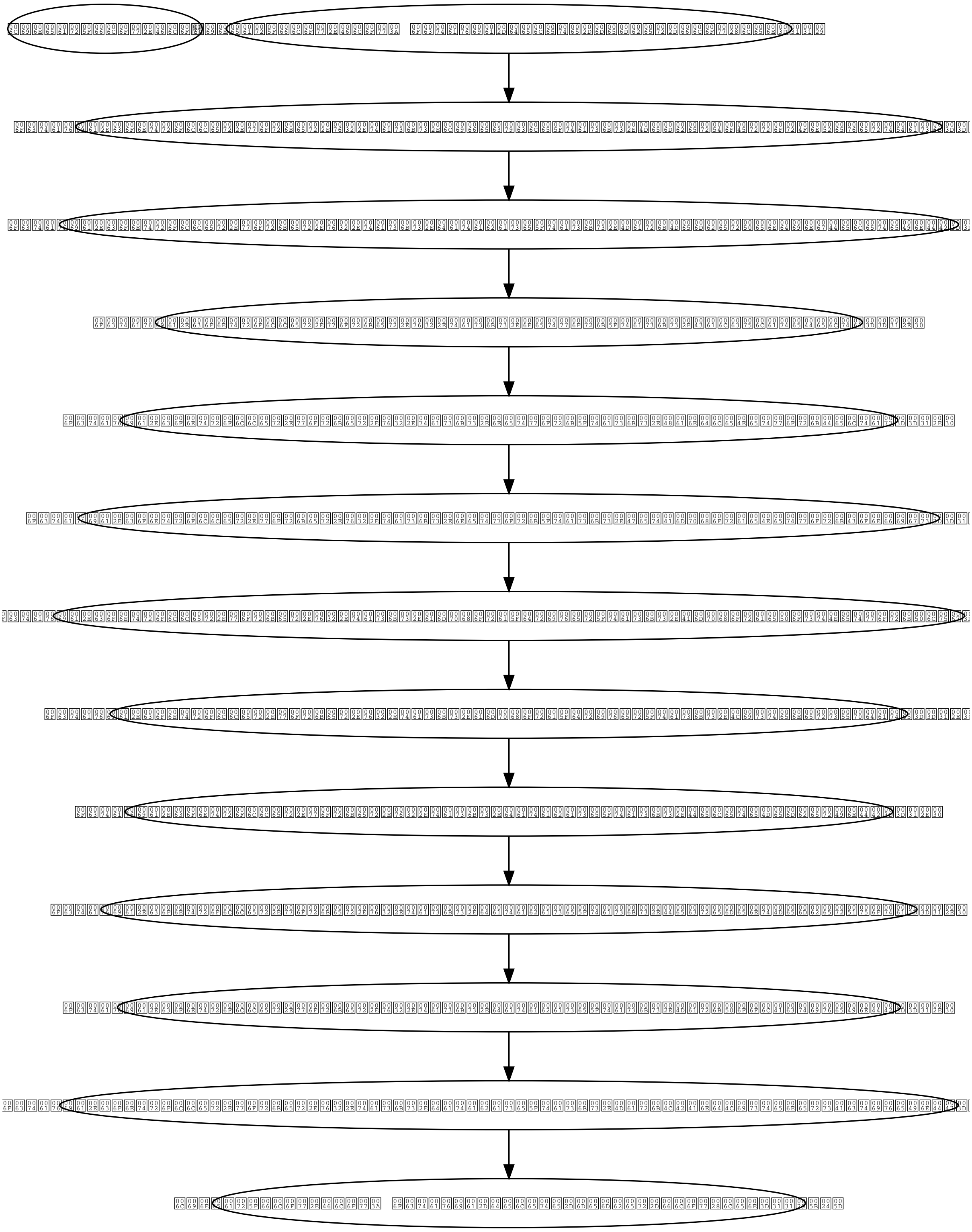
get_batch_update_members_flow



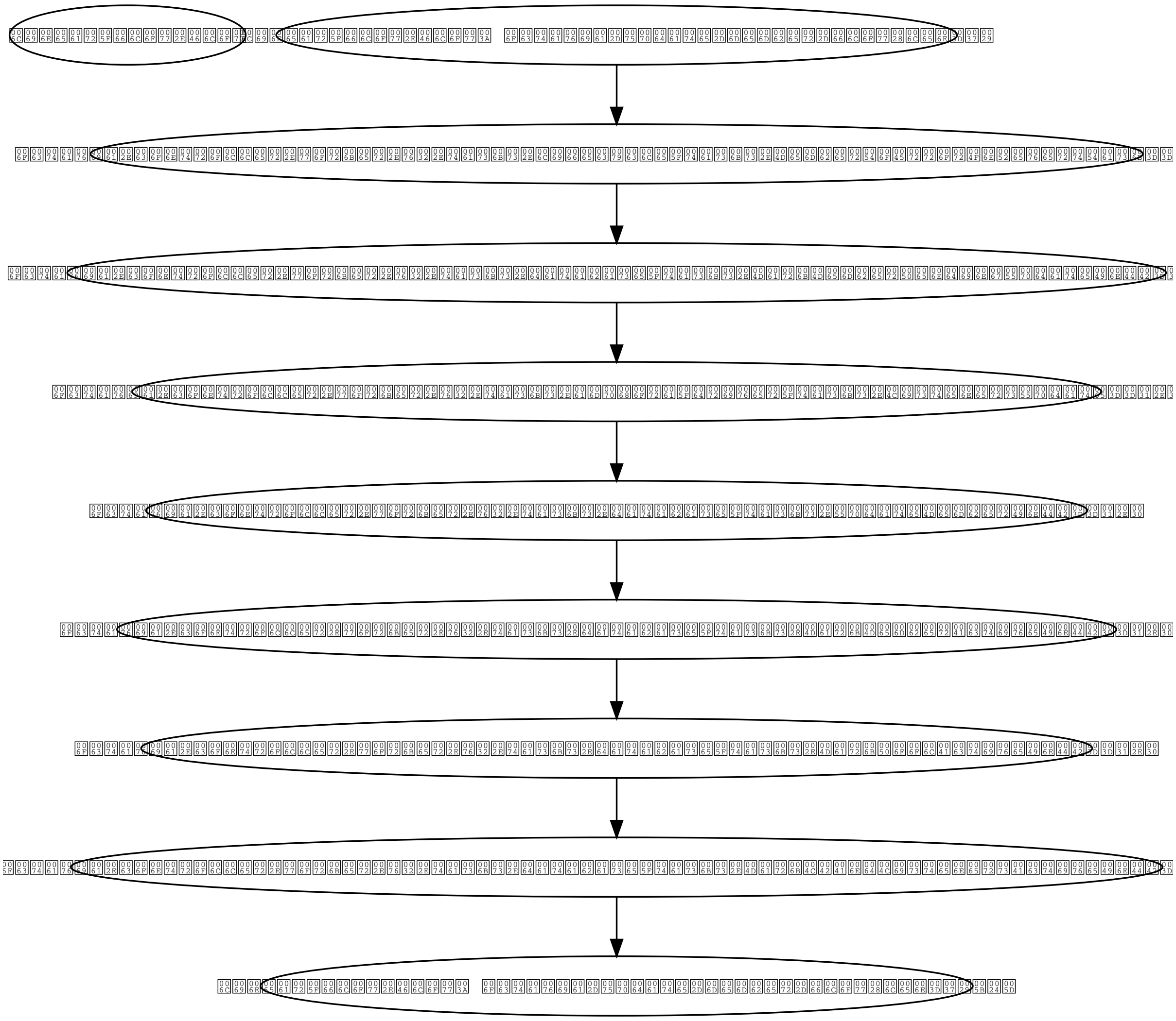
get_create_member_flow



get_delete_member_flow



get_update_member_flow

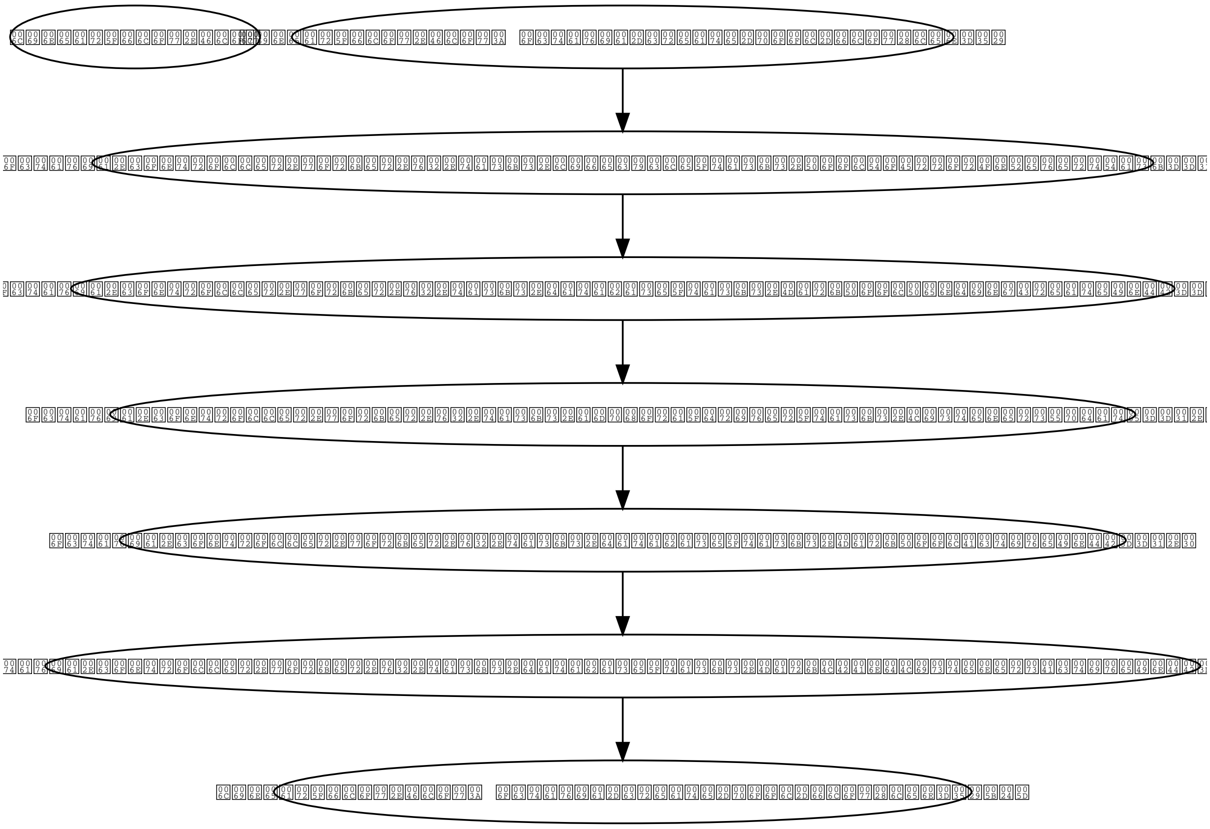


Pool Flows

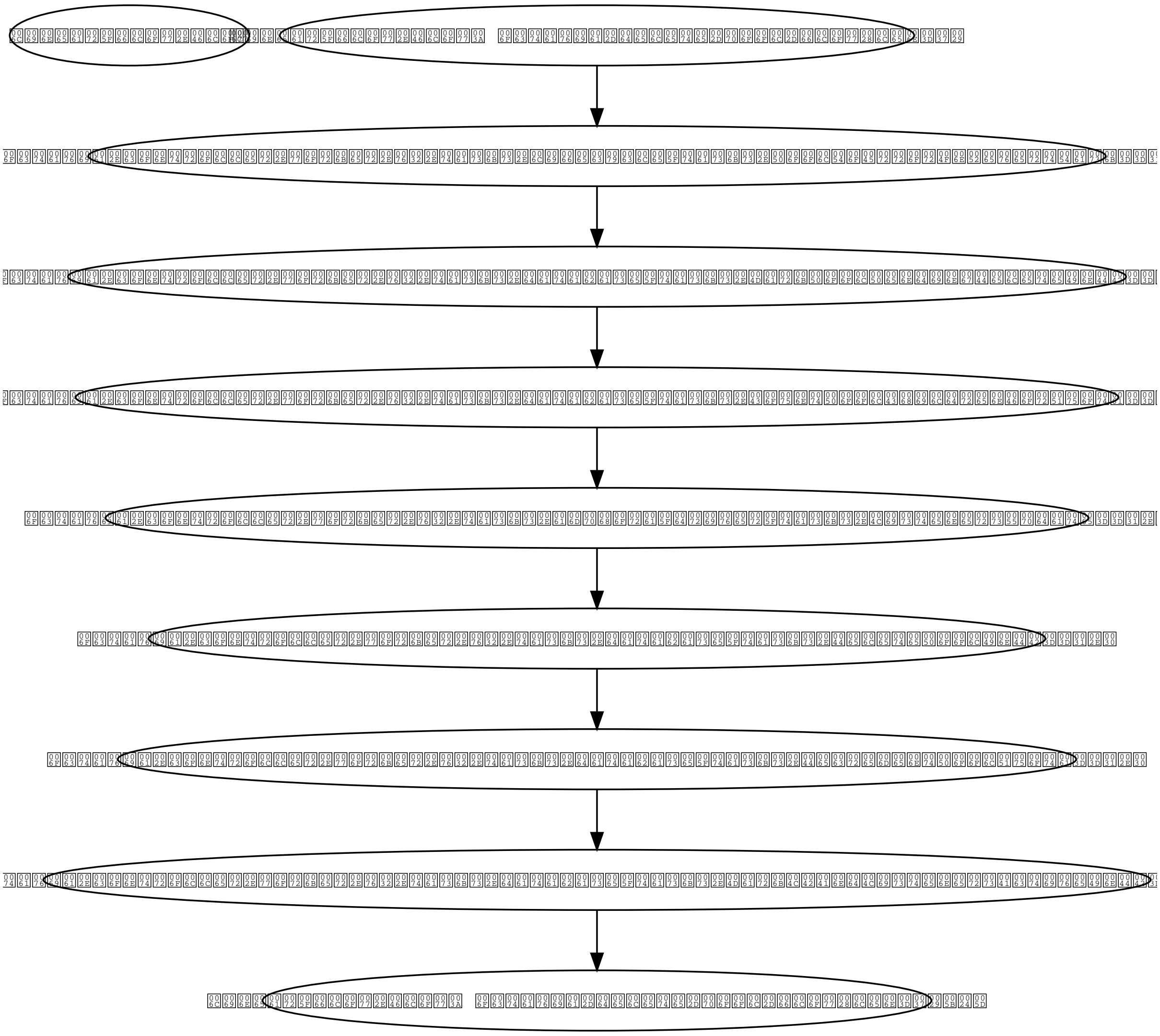
Contents

- *Pool Flows*
  - *get_create_pool_flow*
  - *get_delete_pool_flow*
  - *get_update_pool_flow*

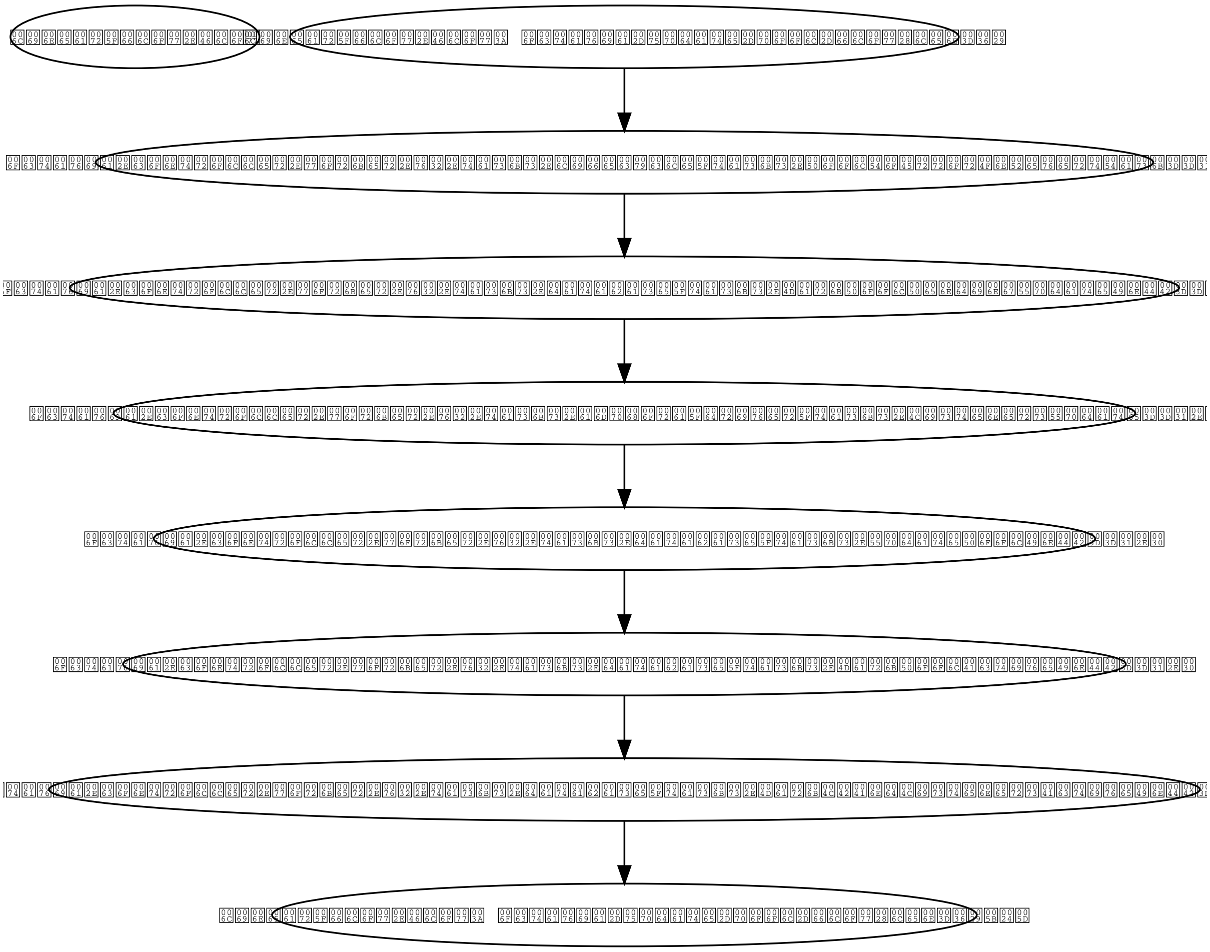
get_create_pool_flow



get_delete_pool_flow



get_update_pool_flow



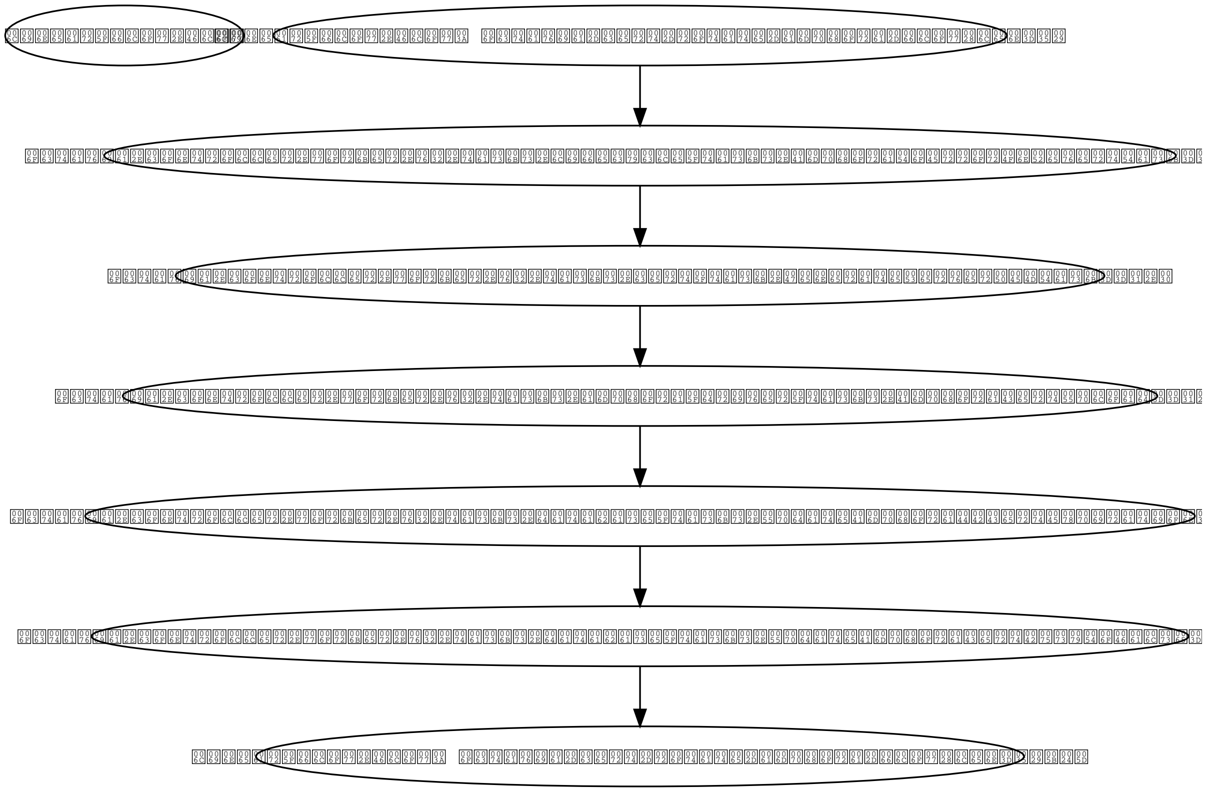
The following are flow diagrams for the **amphora V2** driver.

Amphora Flows

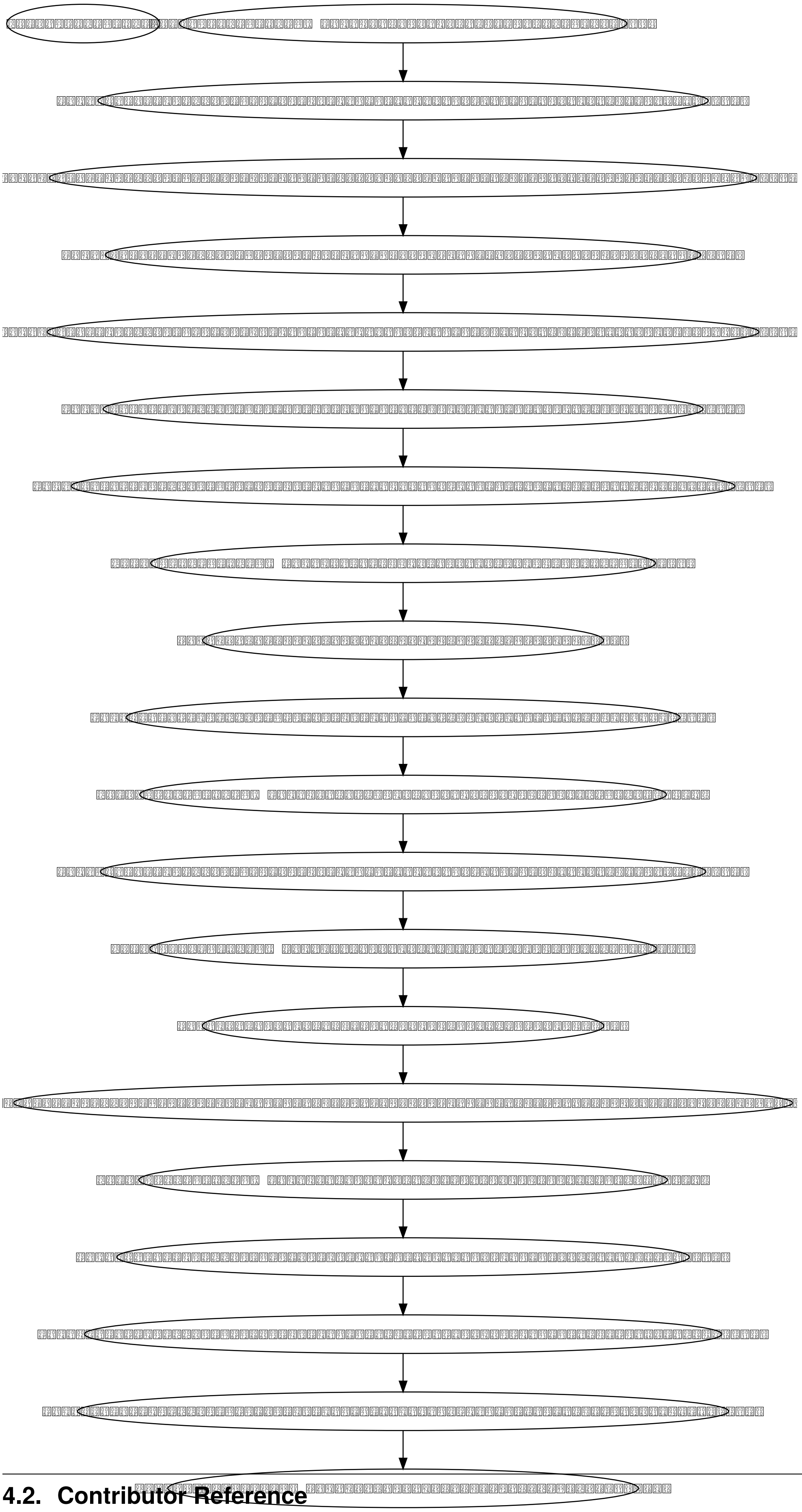
Contents

- *Amphora Flows*
    - *cert_rotate_amphora_flow*
    - *get_create_amphora_flow*
    - *get_failover_amphora_flow*

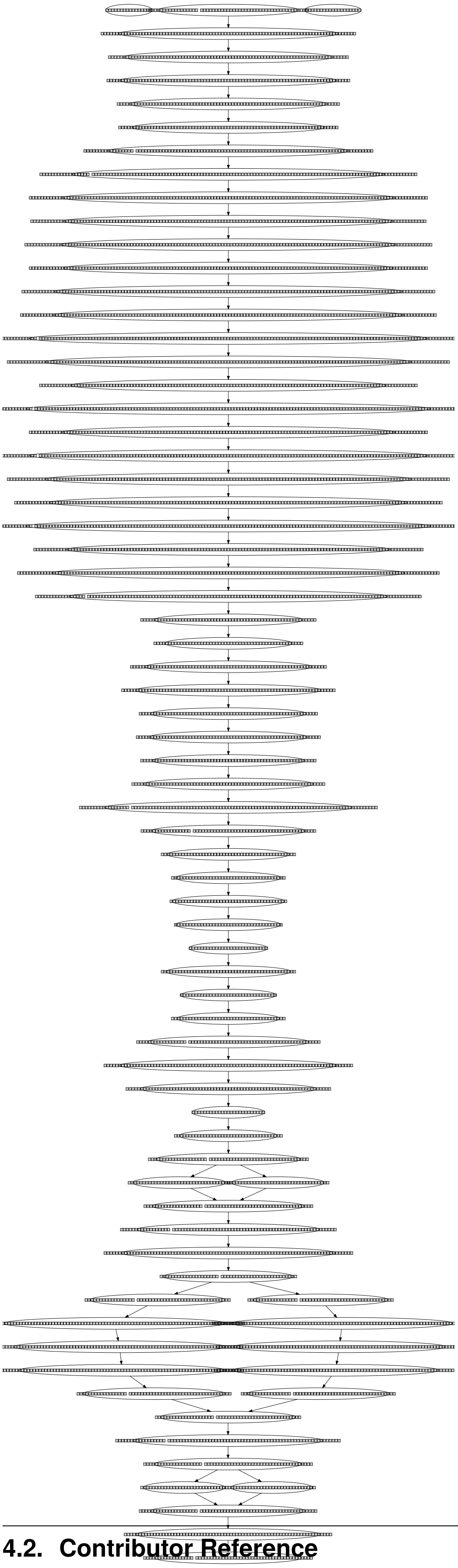
cert_rotate_amphora_flow



get_create_amphora_flow



get_failover_amphora_flow

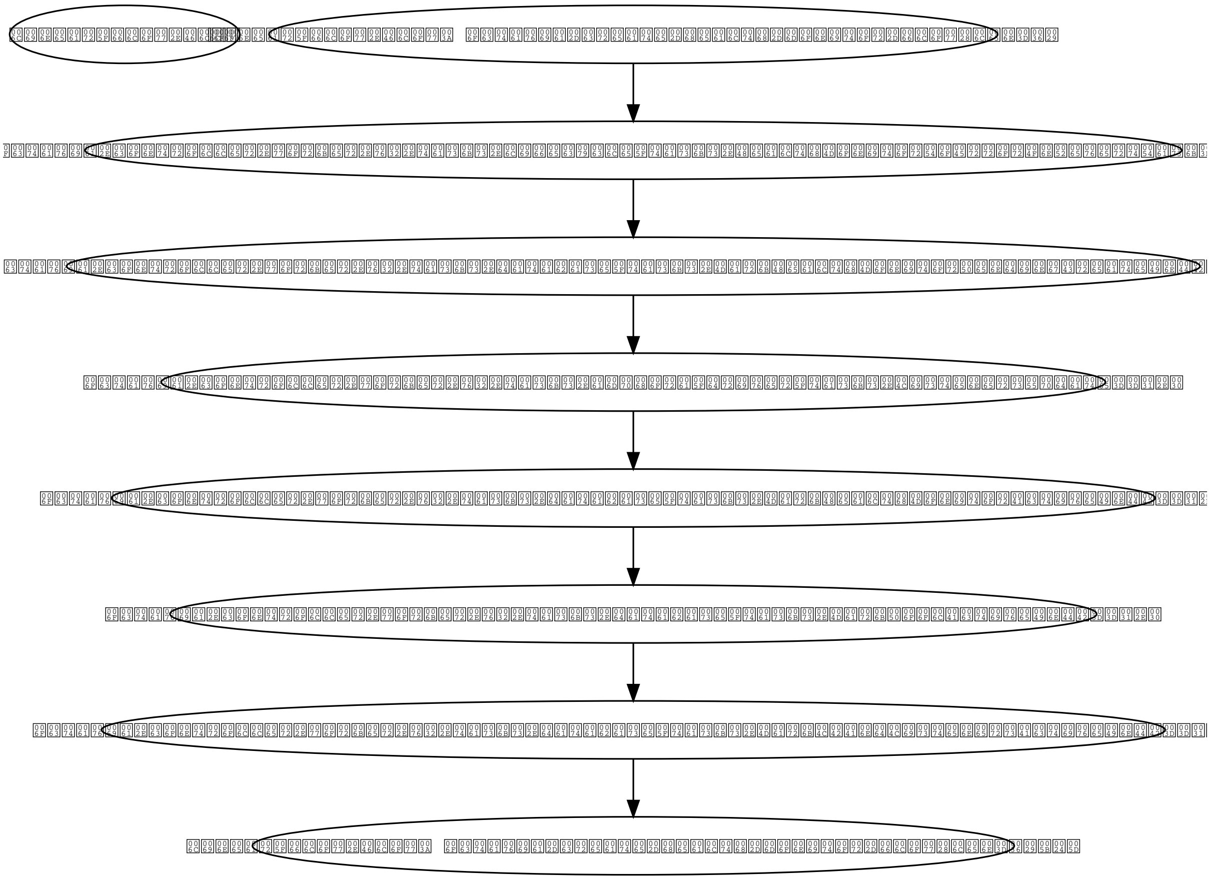


Health Monitor Flows

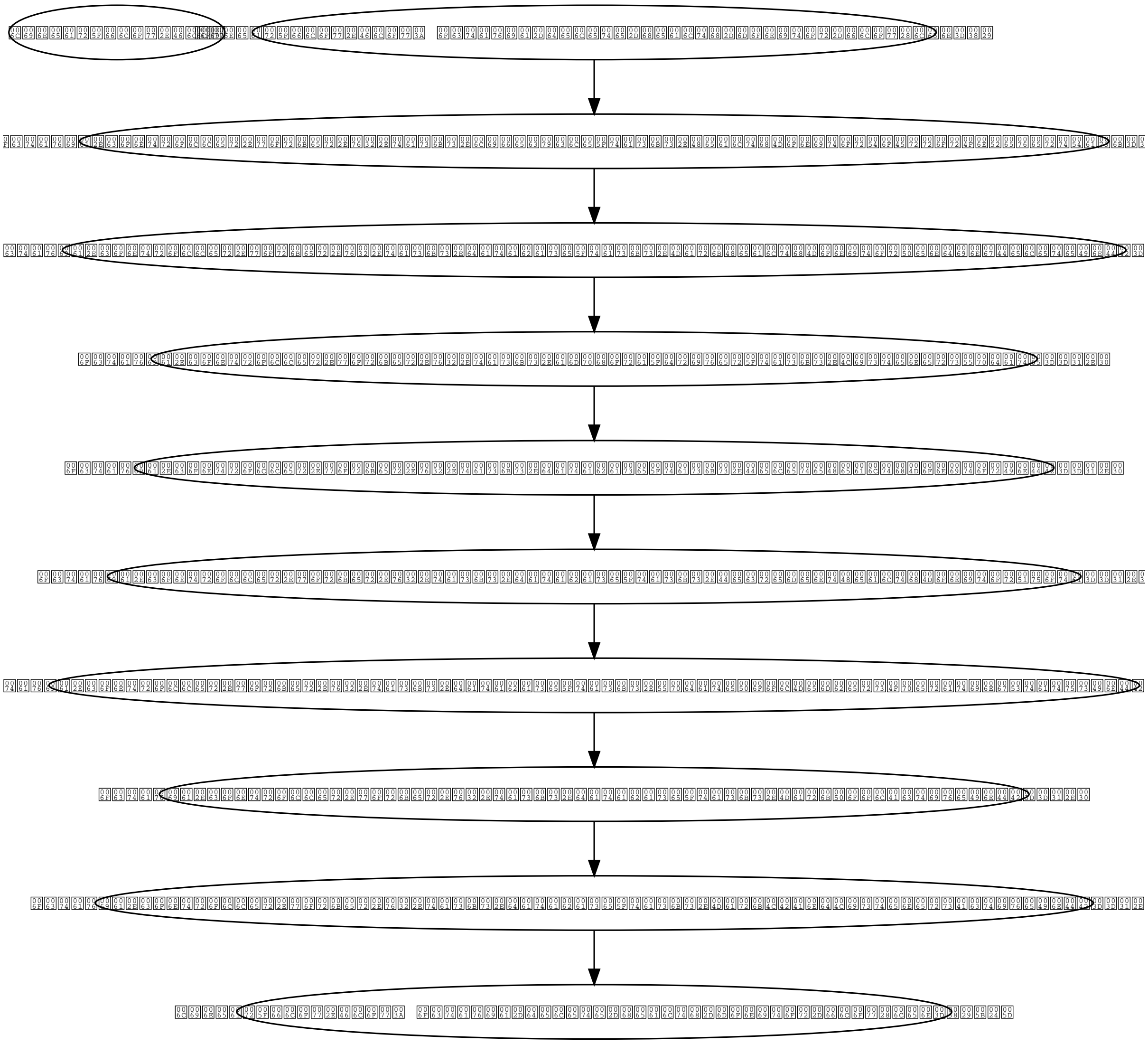
Contents

- *Health Monitor Flows*
  - *get_create_health_monitor_flow*
  - *get_delete_health_monitor_flow*
  - *get_update_health_monitor_flow*

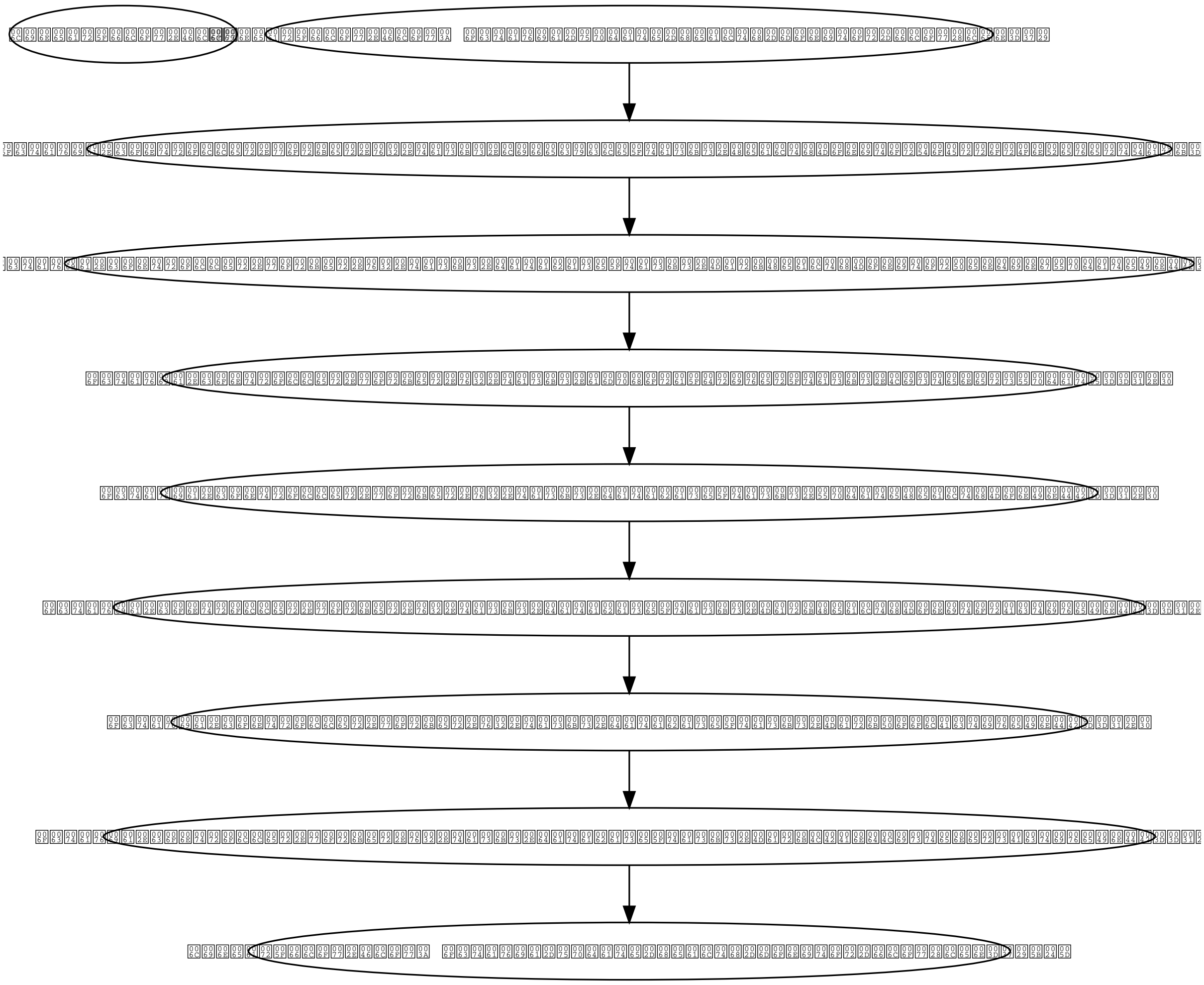
get_create_health_monitor_flow



get_delete_health_monitor_flow



get_update_health_monitor_flow

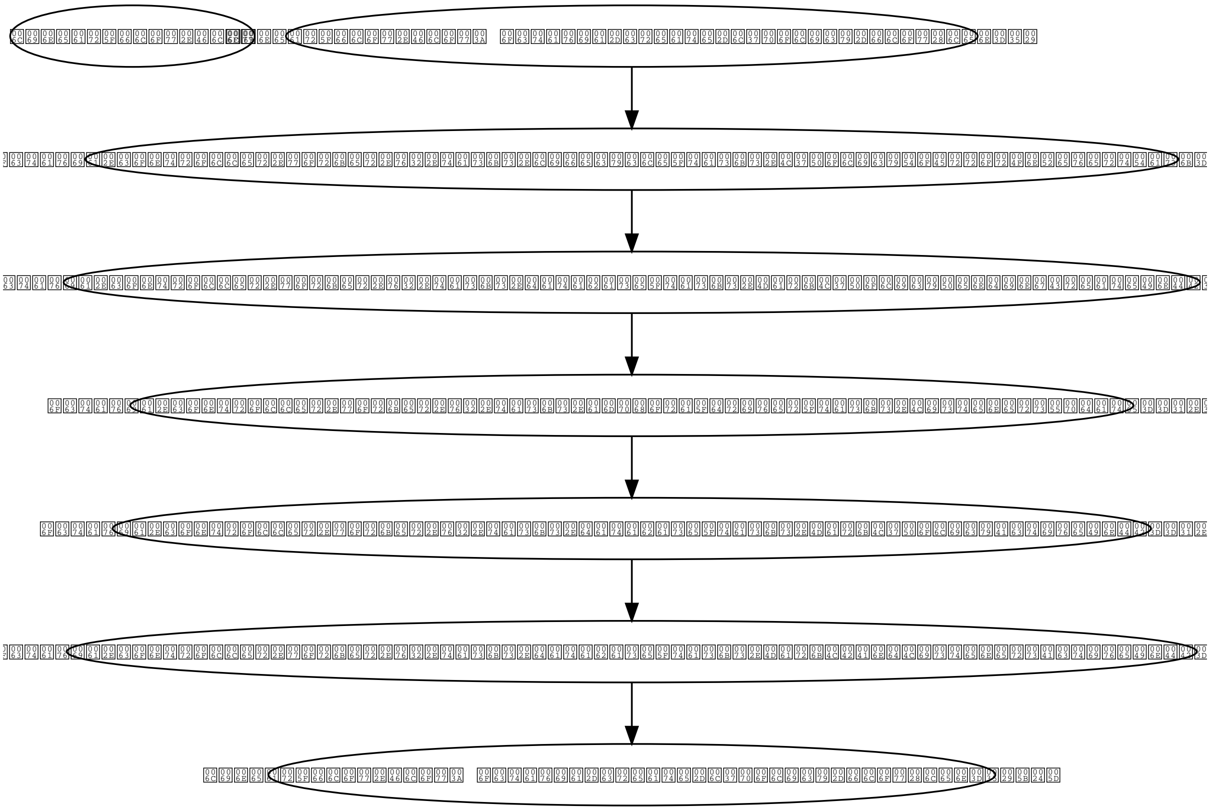


Layer 7 Policy Flows

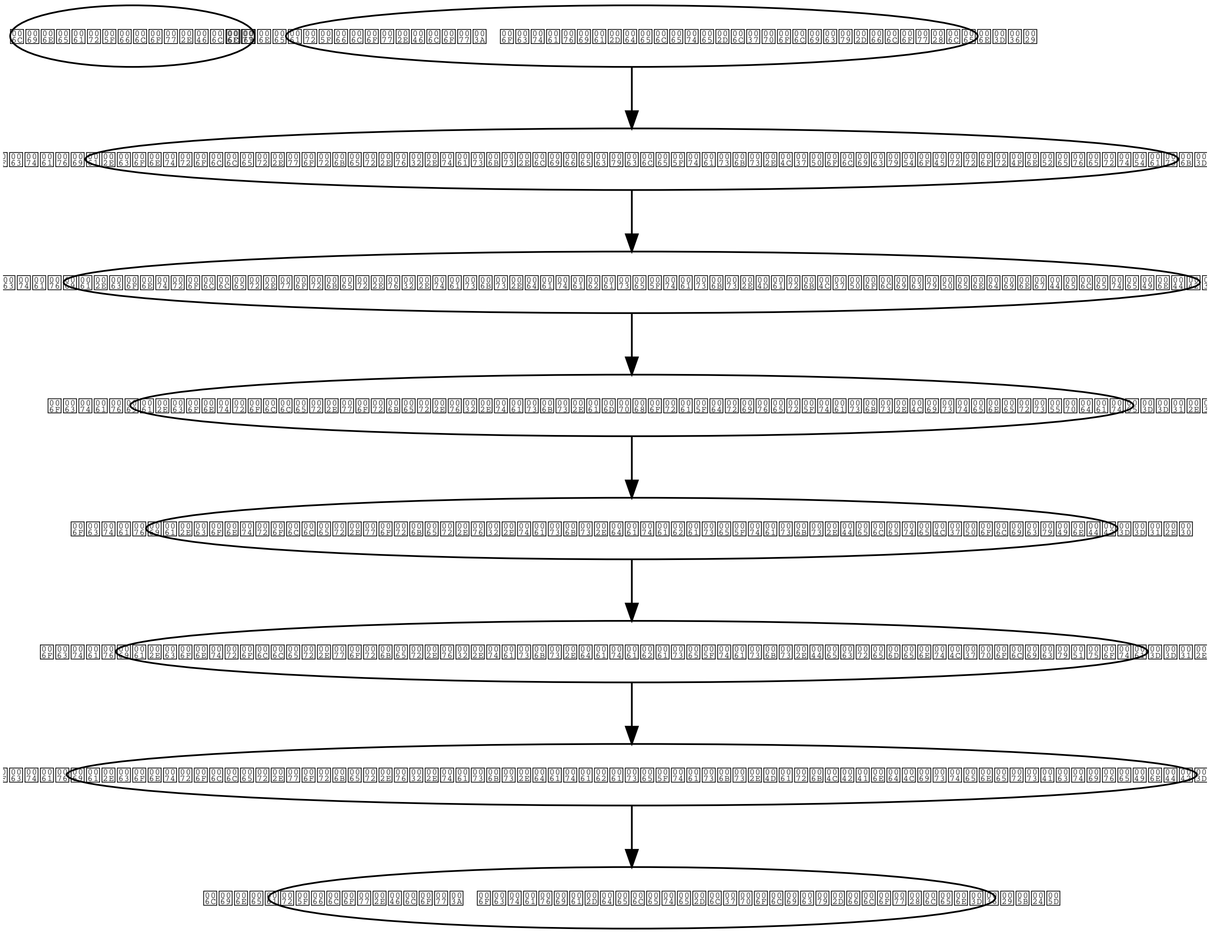
Contents

- *Layer 7 Policy Flows*
  - *get_create_l7policy_flow*
  - *get_delete_l7policy_flow*
  - *get_update_l7policy_flow*

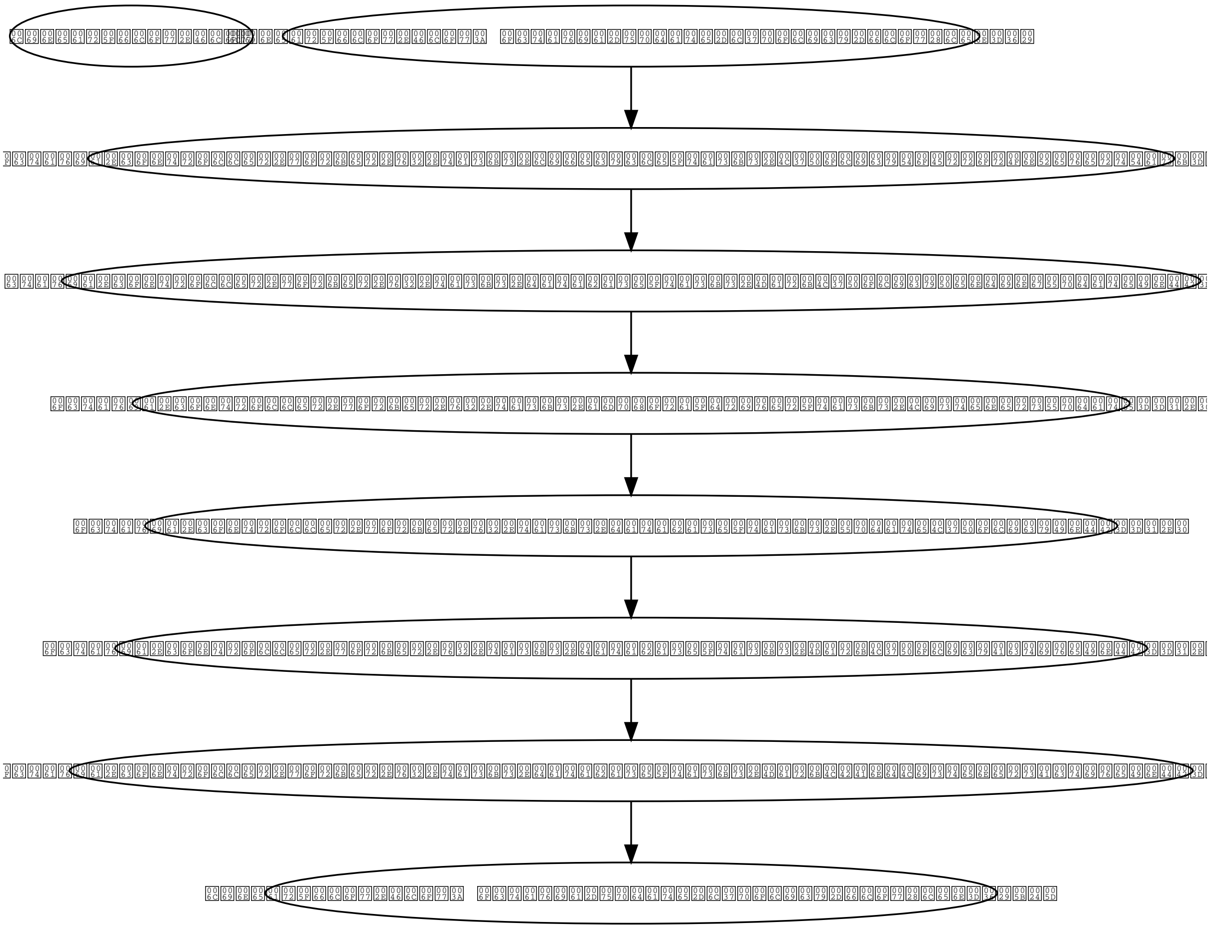
get_create_l7policy_flow



get_delete_l7policy_flow



get_update_l7policy_flow

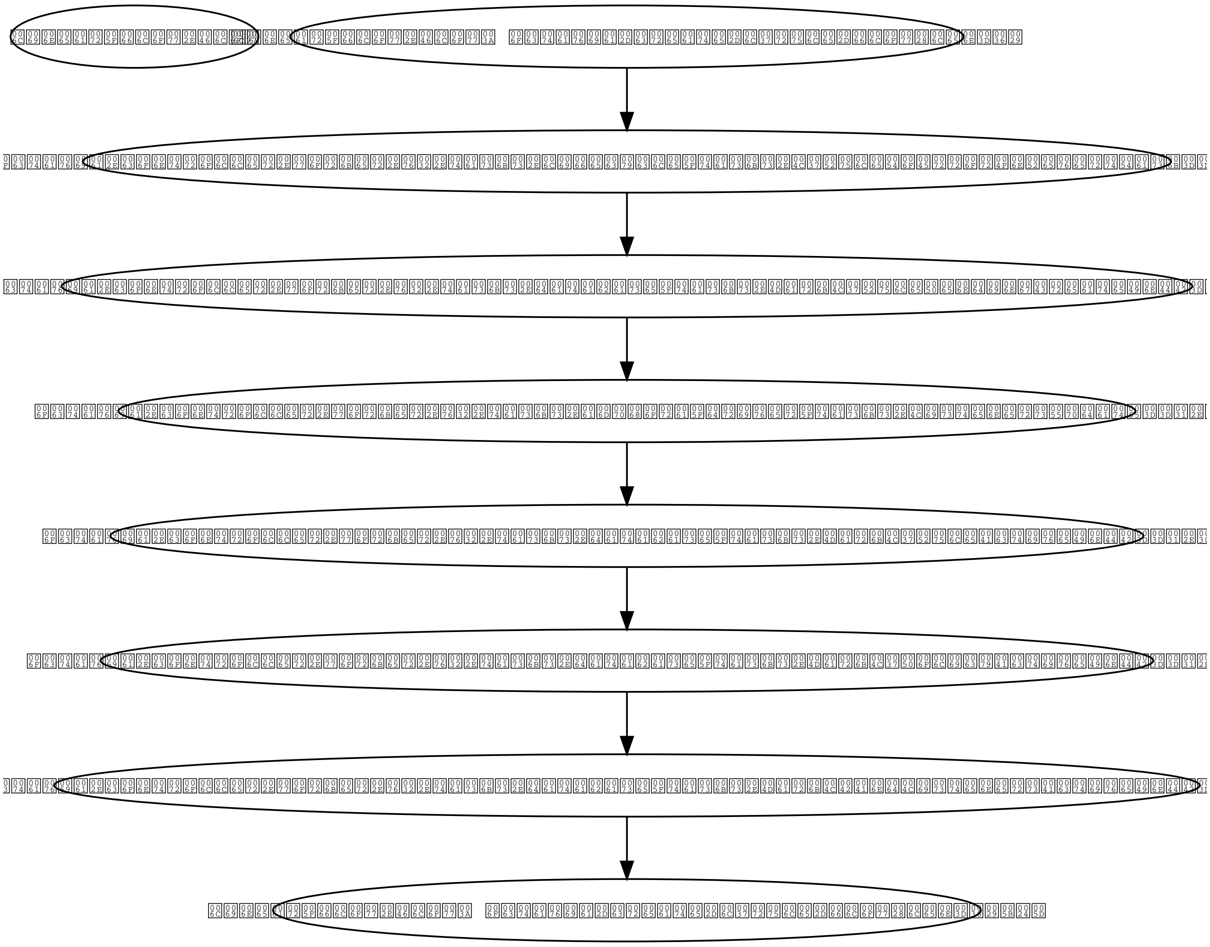


Layer 7 Rule Flows

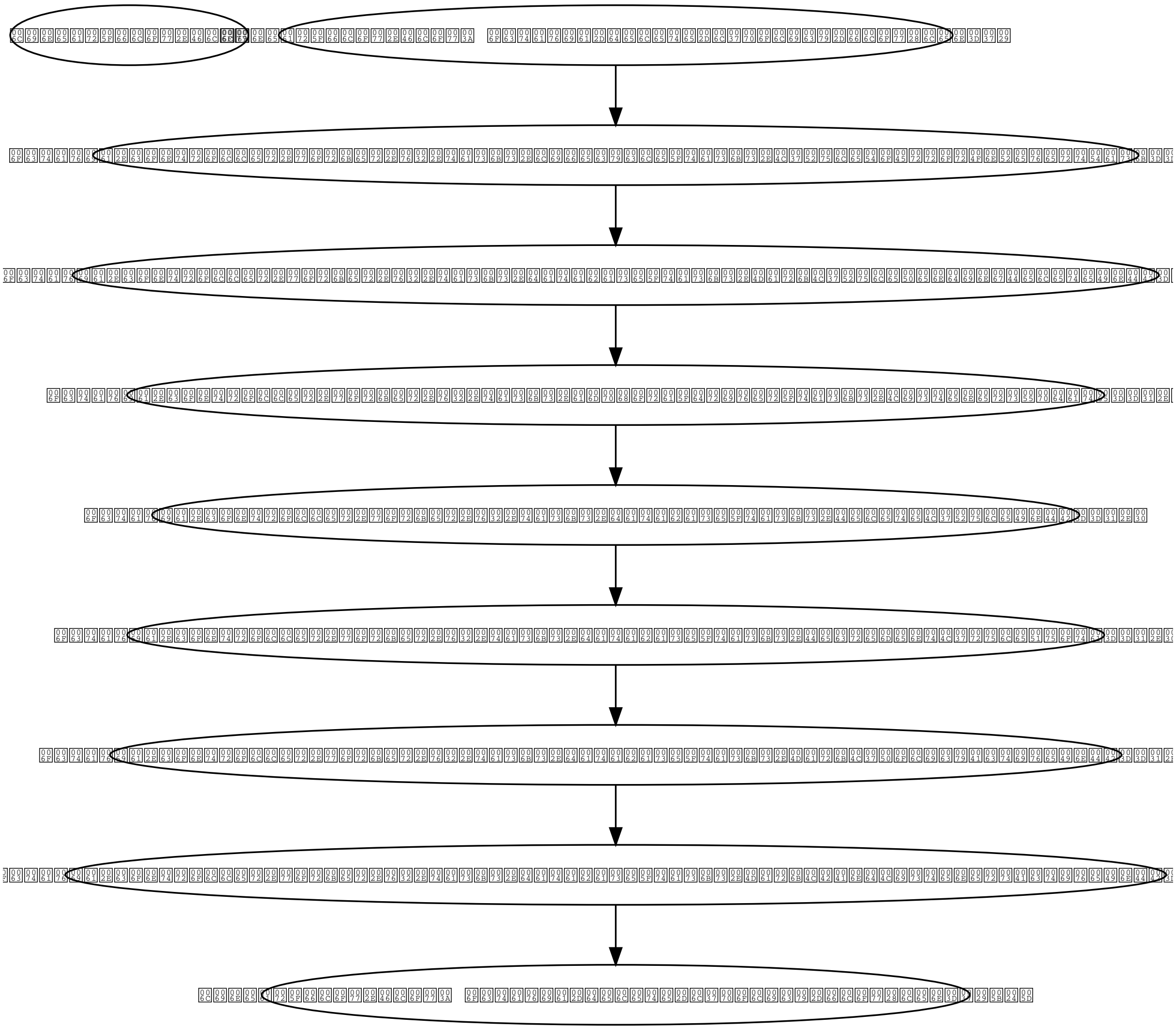
Contents

- *Layer 7 Rule Flows*
  - *get_create_l7rule_flow*
  - *get_delete_l7rule_flow*
  - *get_update_l7rule_flow*

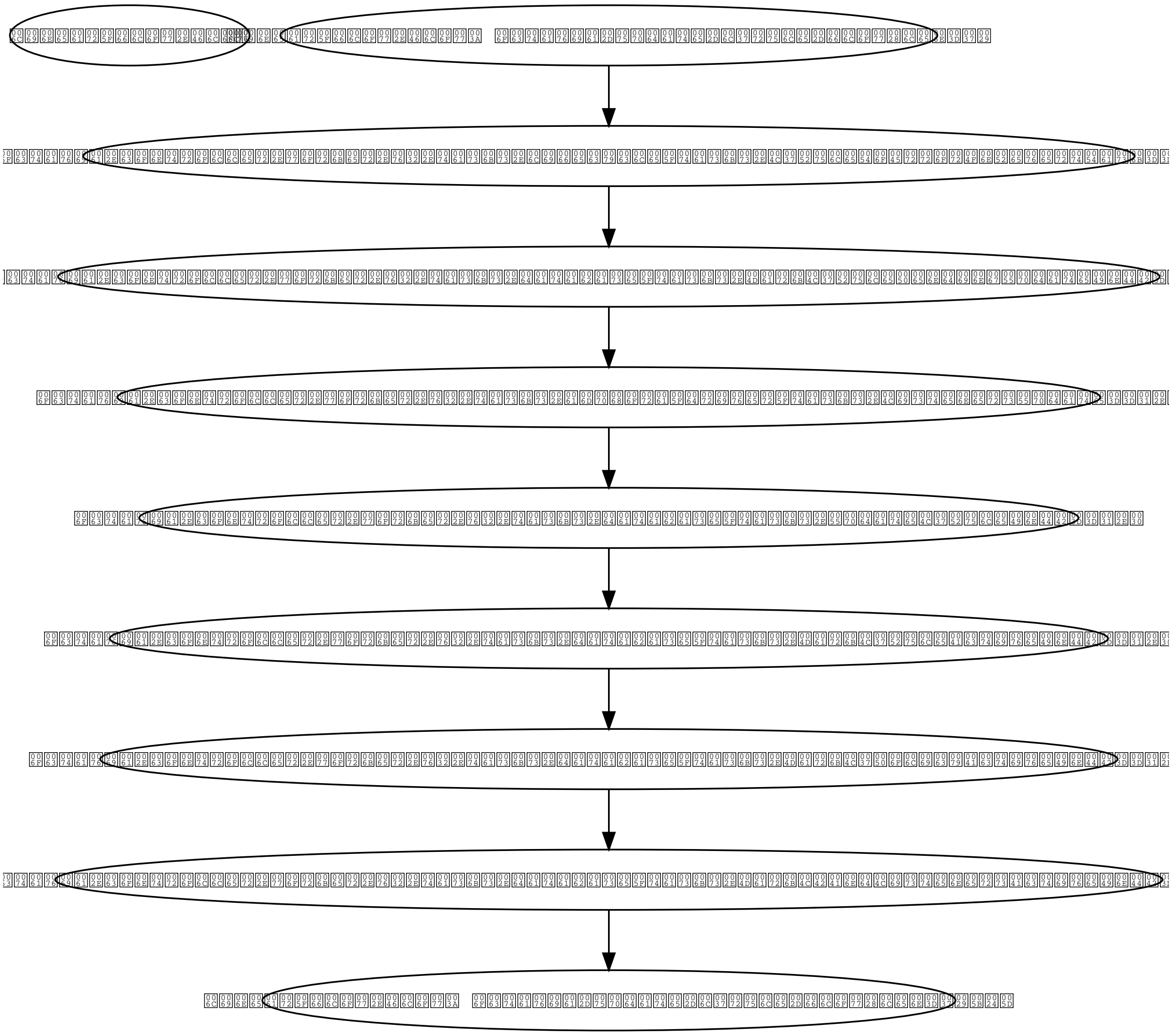
get_create_l7rule_flow



get_delete_l7rule_flow



get_update_l7rule_flow

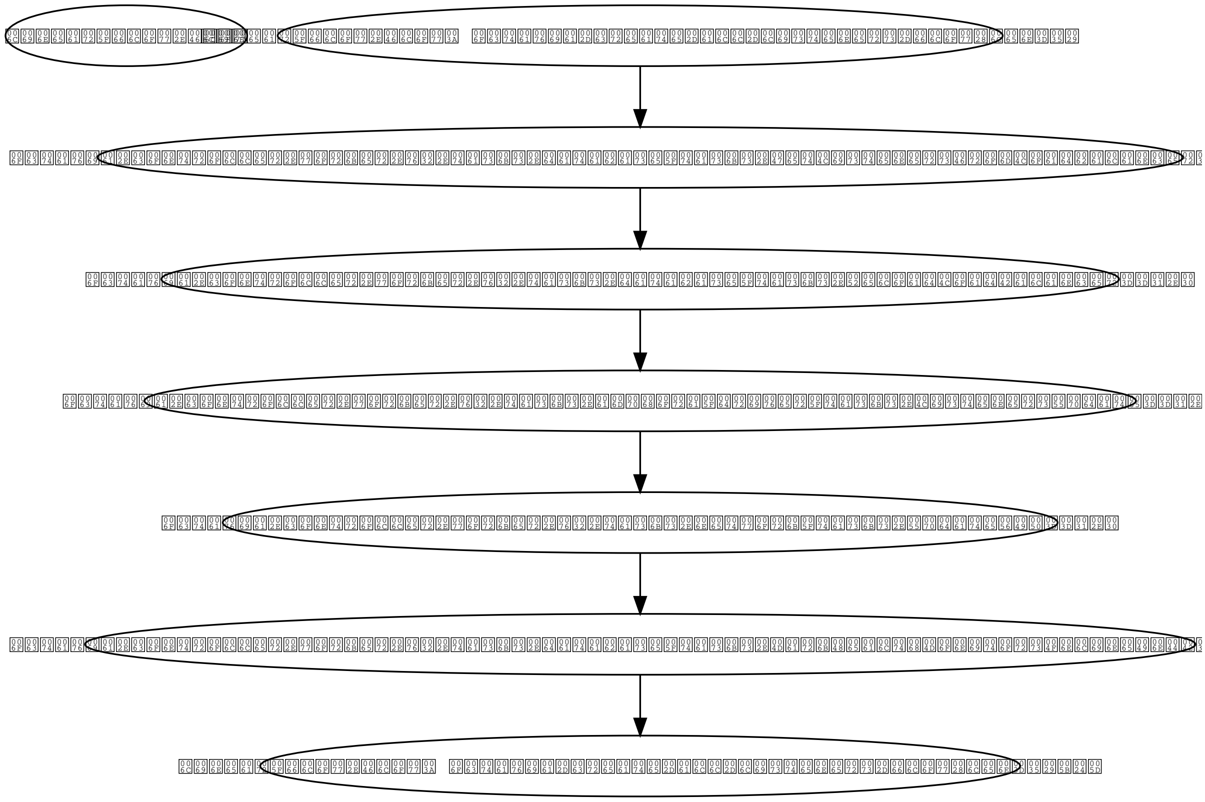


Listener Flows

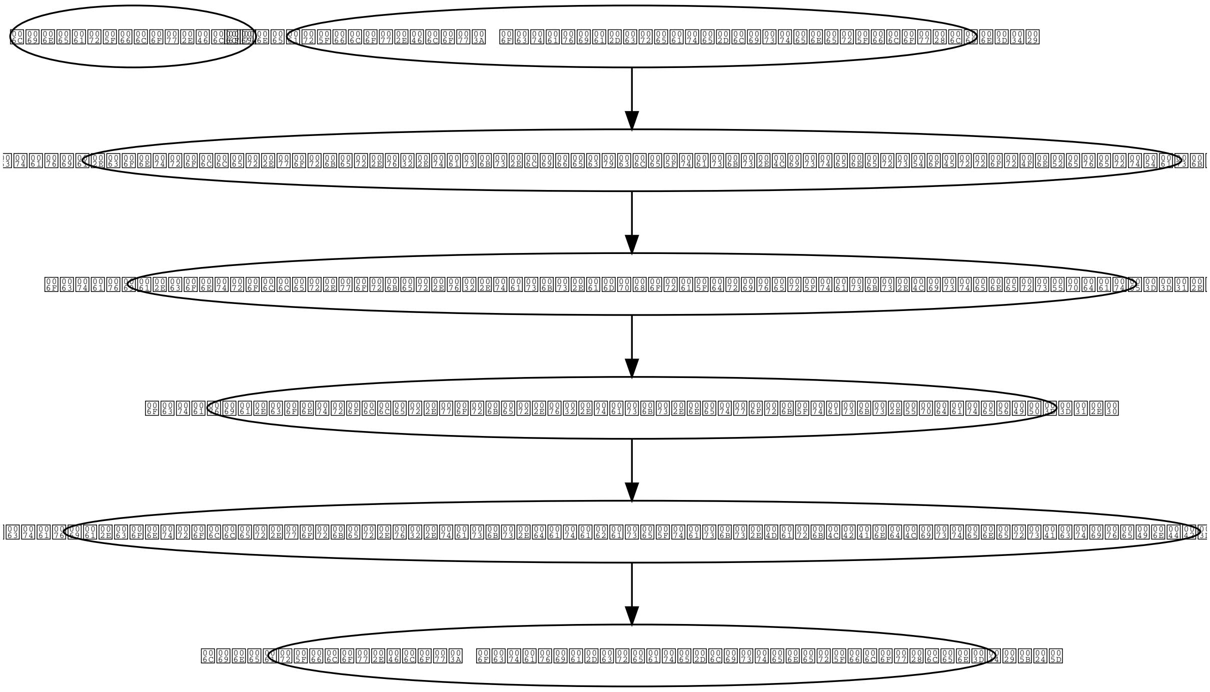
Contents

- *Listener Flows*
  - *get_create_all_listeners_flow*
  - *get_create_listener_flow*
  - *get_delete_listener_flow*
  - *get_update_listener_flow*

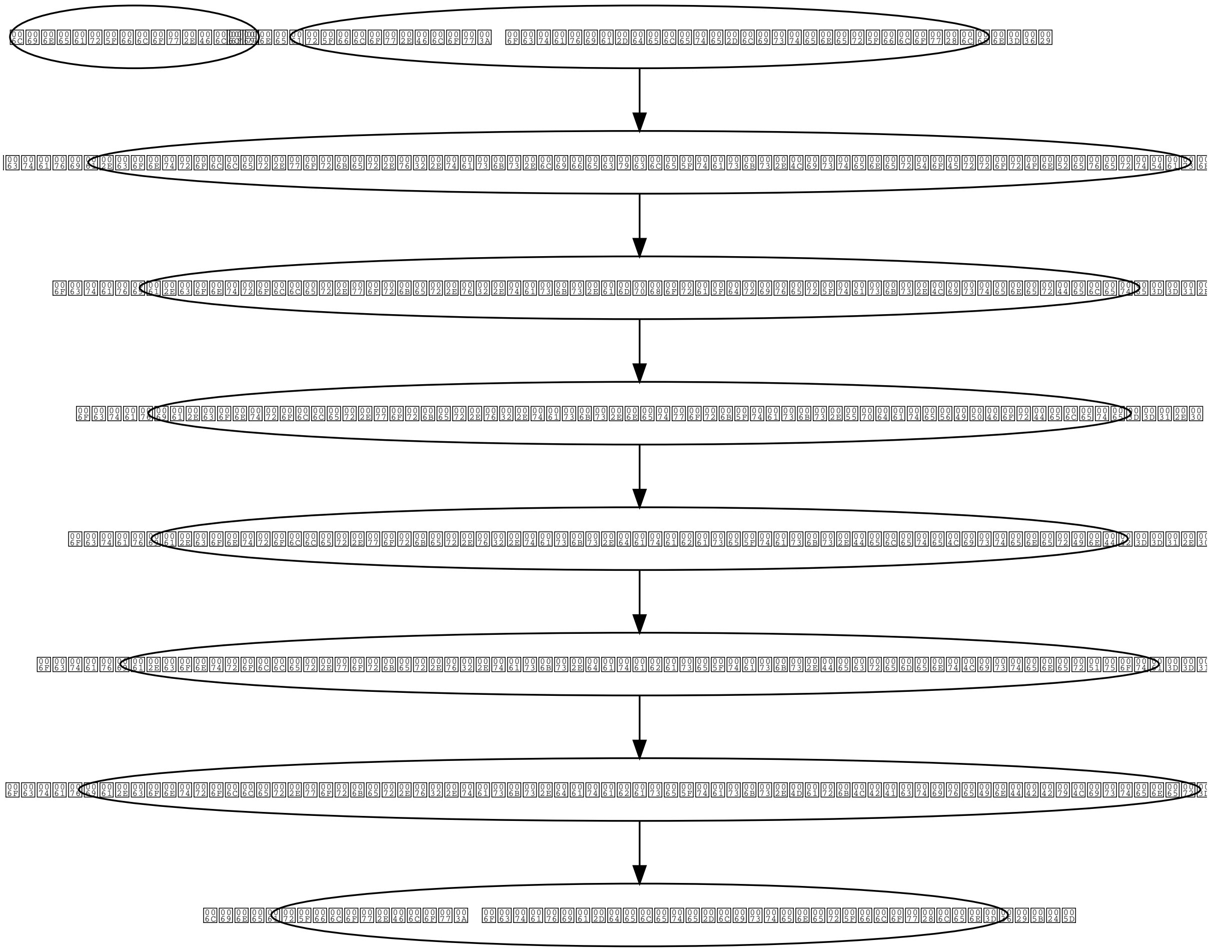
get_create_all_listeners_flow



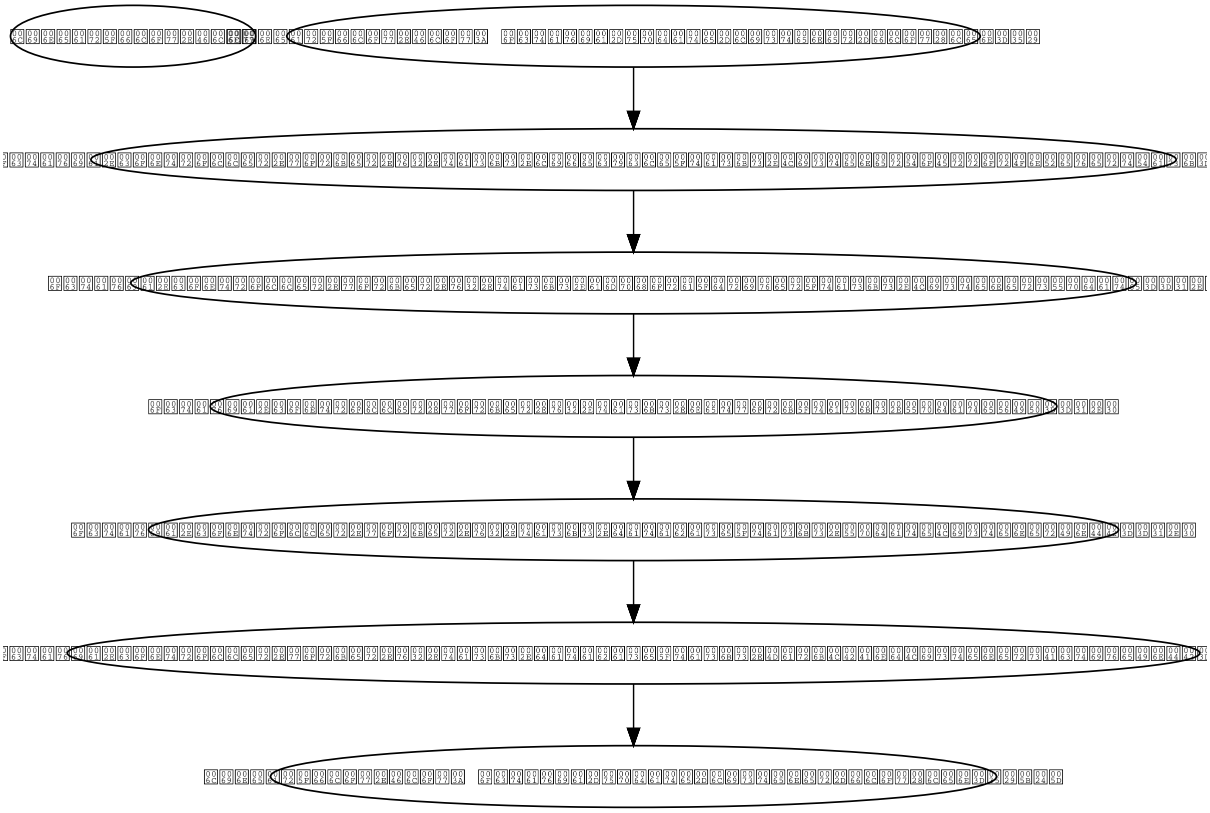
get_create_listener_flow



get_delete_listener_flow



get_update_listener_flow

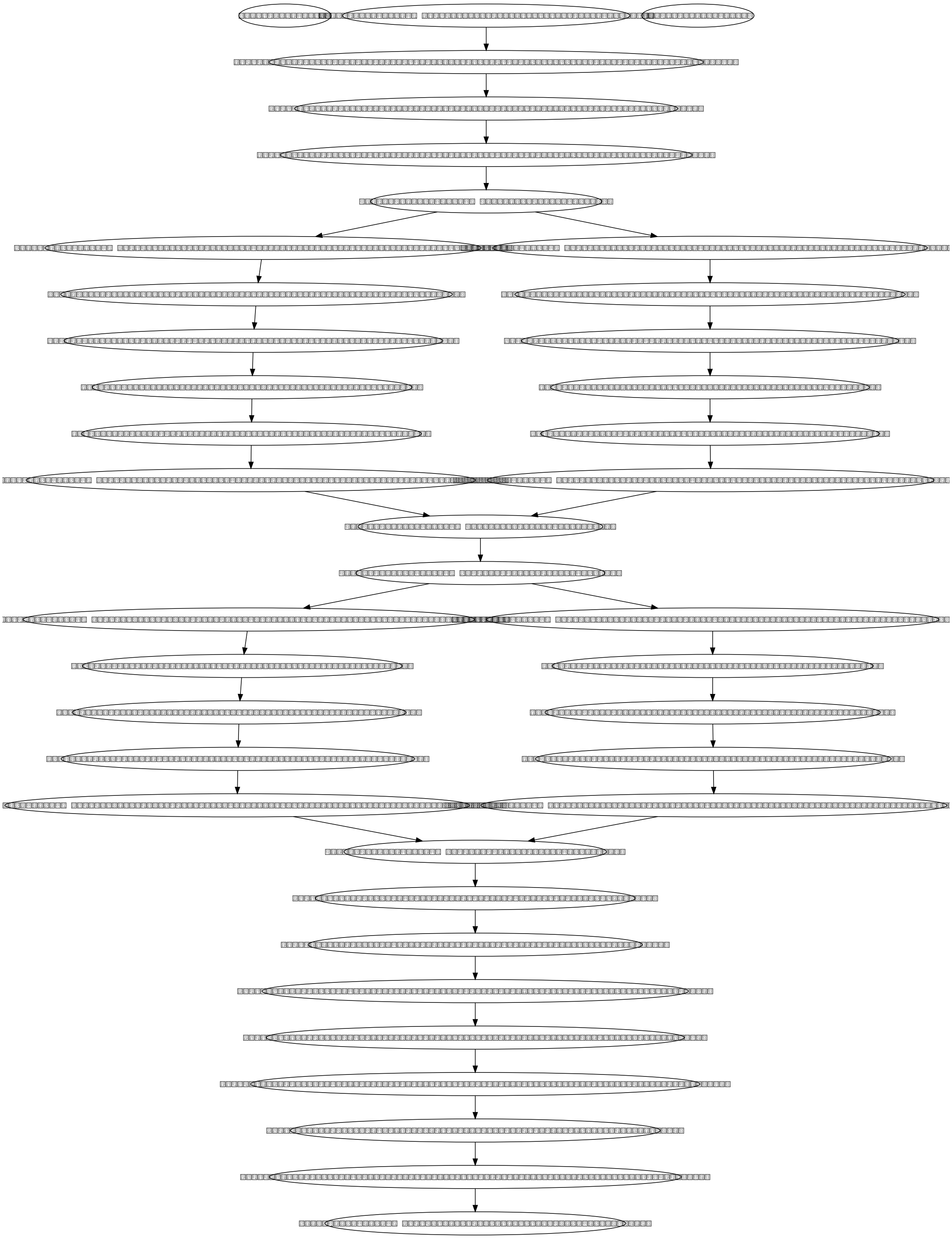


Load Balancer Flows

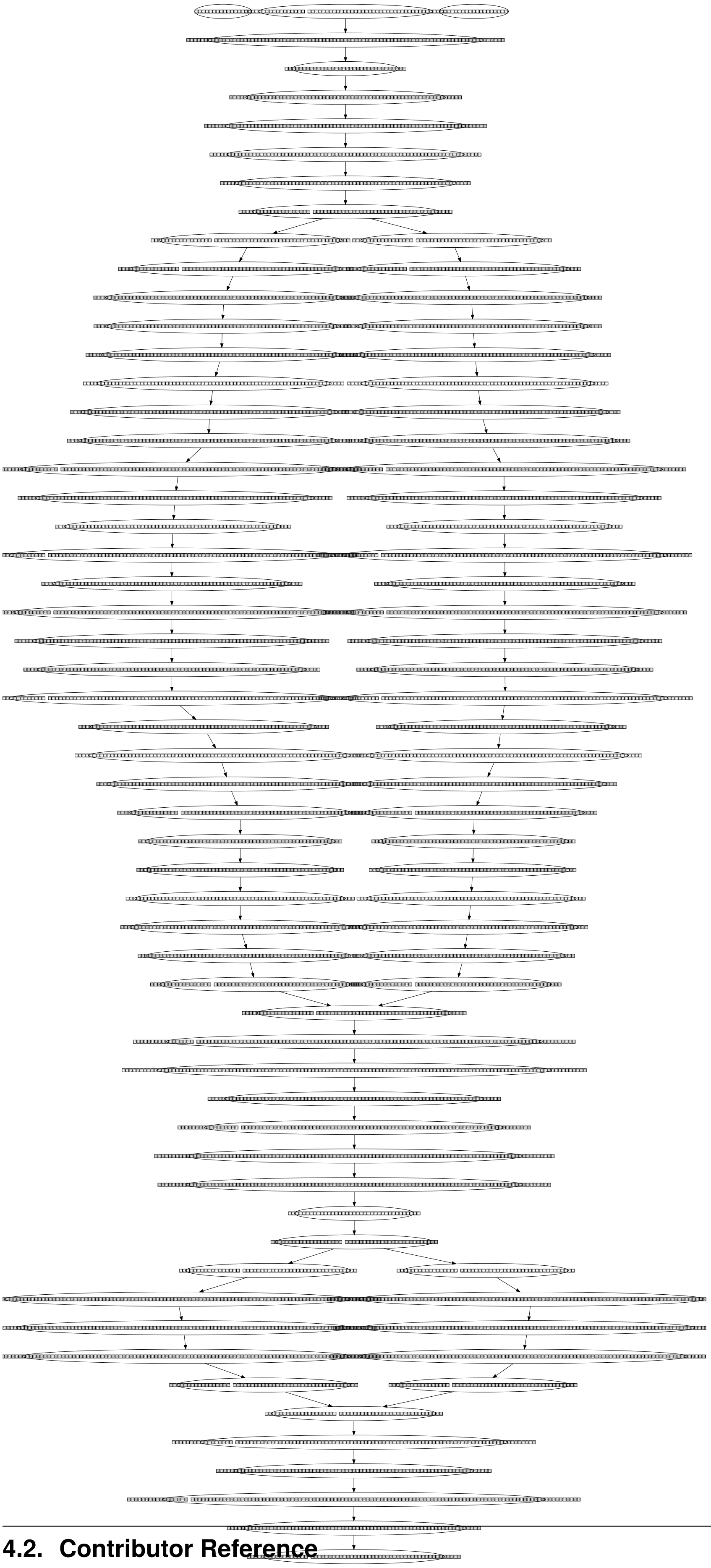
Contents

- *Load Balancer Flows*
  - *get_cascade_delete_load_balancer_flow*
  - *get_create_load_balancer_flow*
  - *get_delete_load_balancer_flow*
  - *get_failover_LB_flow*
  - *get_update_load_balancer_flow*

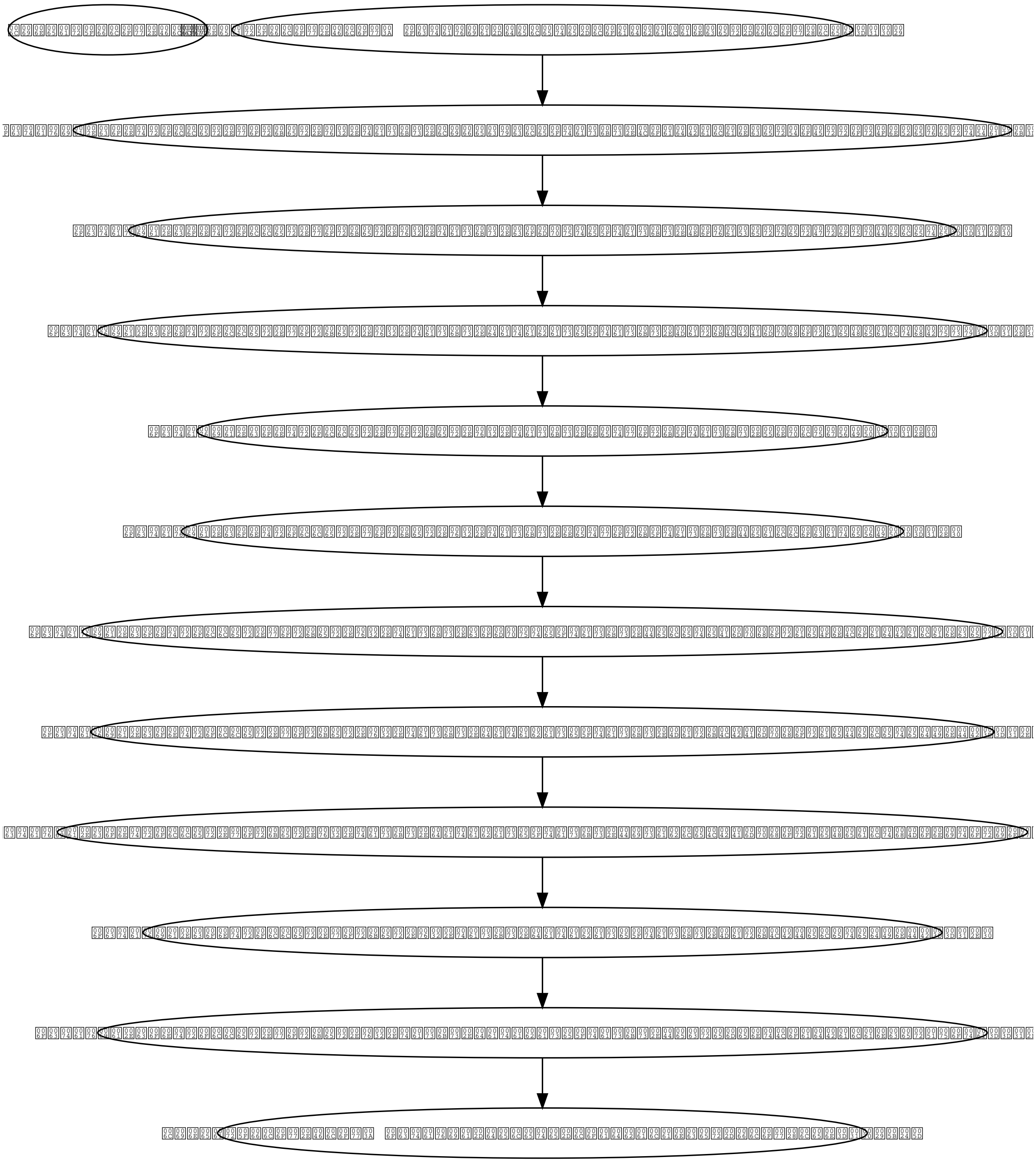
get_cascade_delete_load_balancer_flow



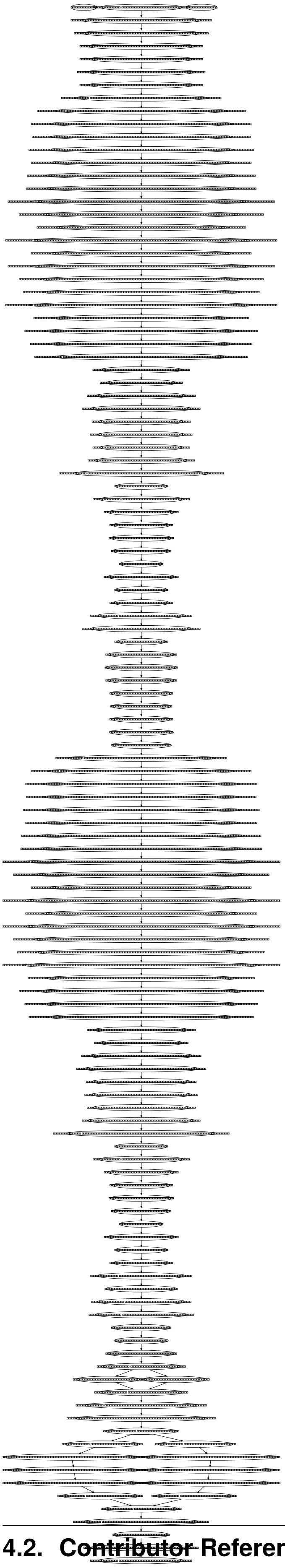
get_create_load_balancer_flow



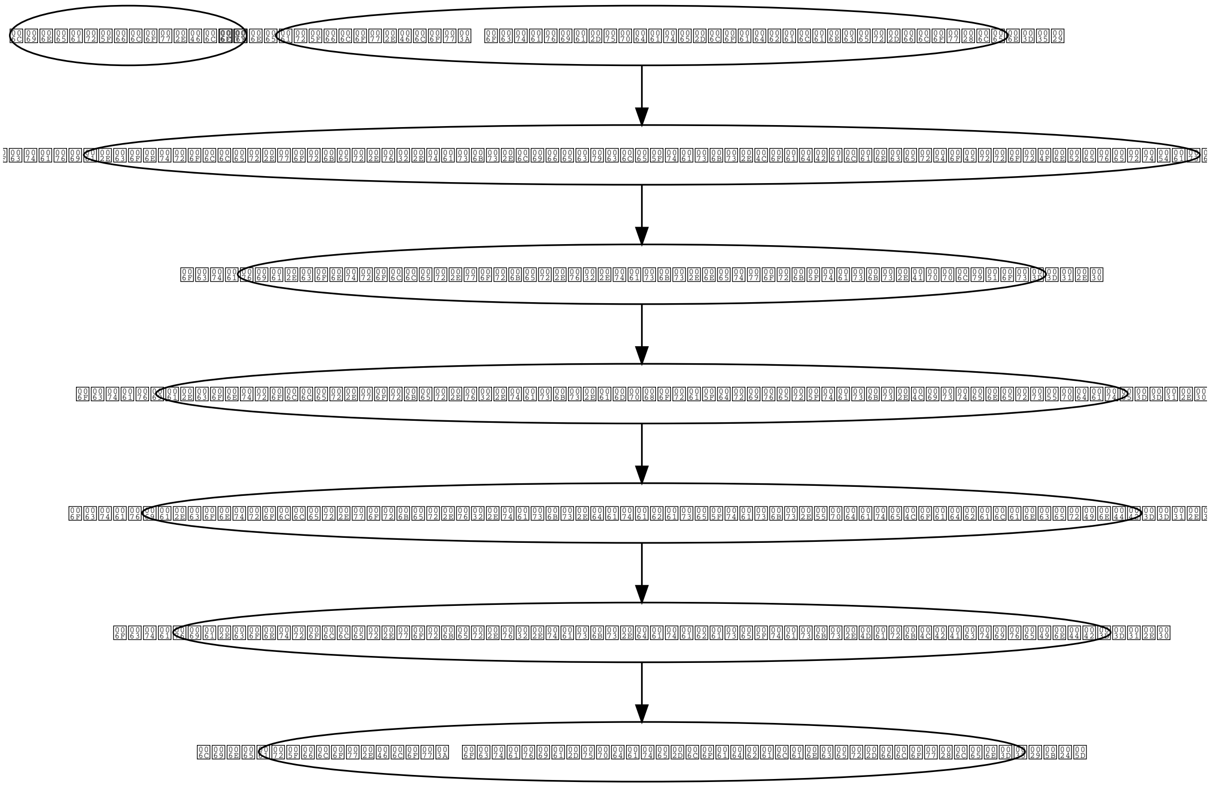
get_delete_load_balancer_flow



get_failover_LB_flow



get_update_load_balancer_flow

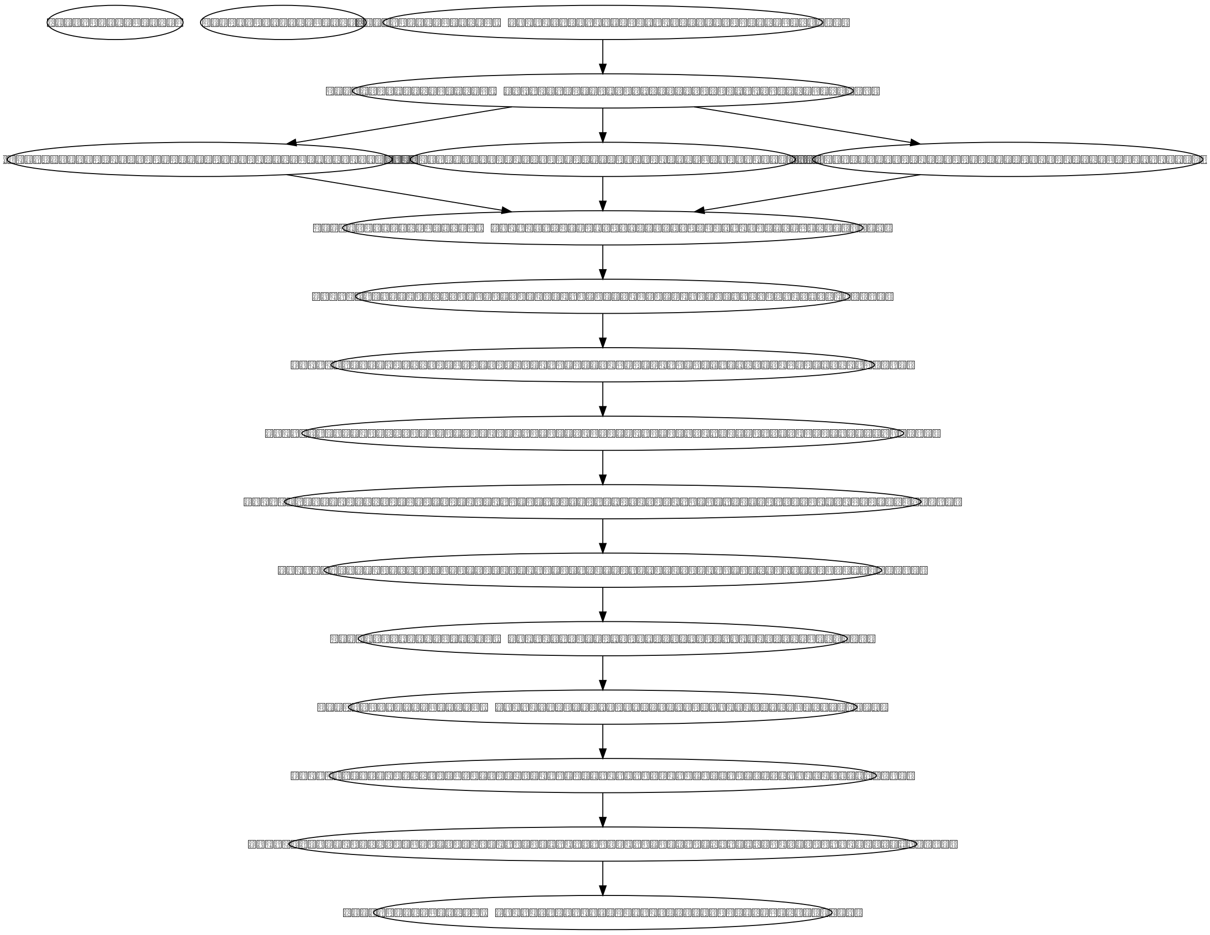


Member Flows

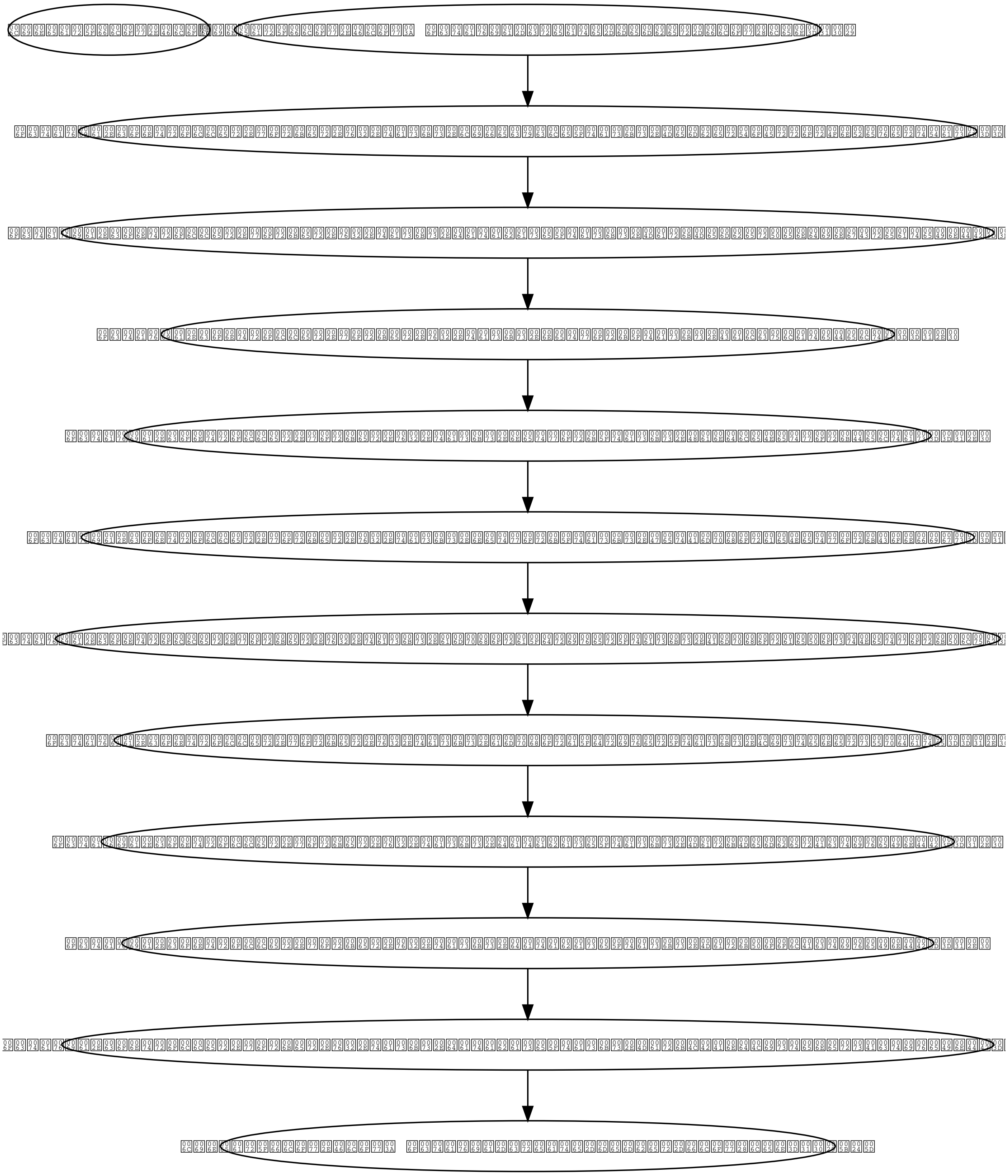
Contents

- *Member Flows*
  - *get_batch_update_members_flow*
  - *get_create_member_flow*
  - *get_delete_member_flow*
  - *get_update_member_flow*

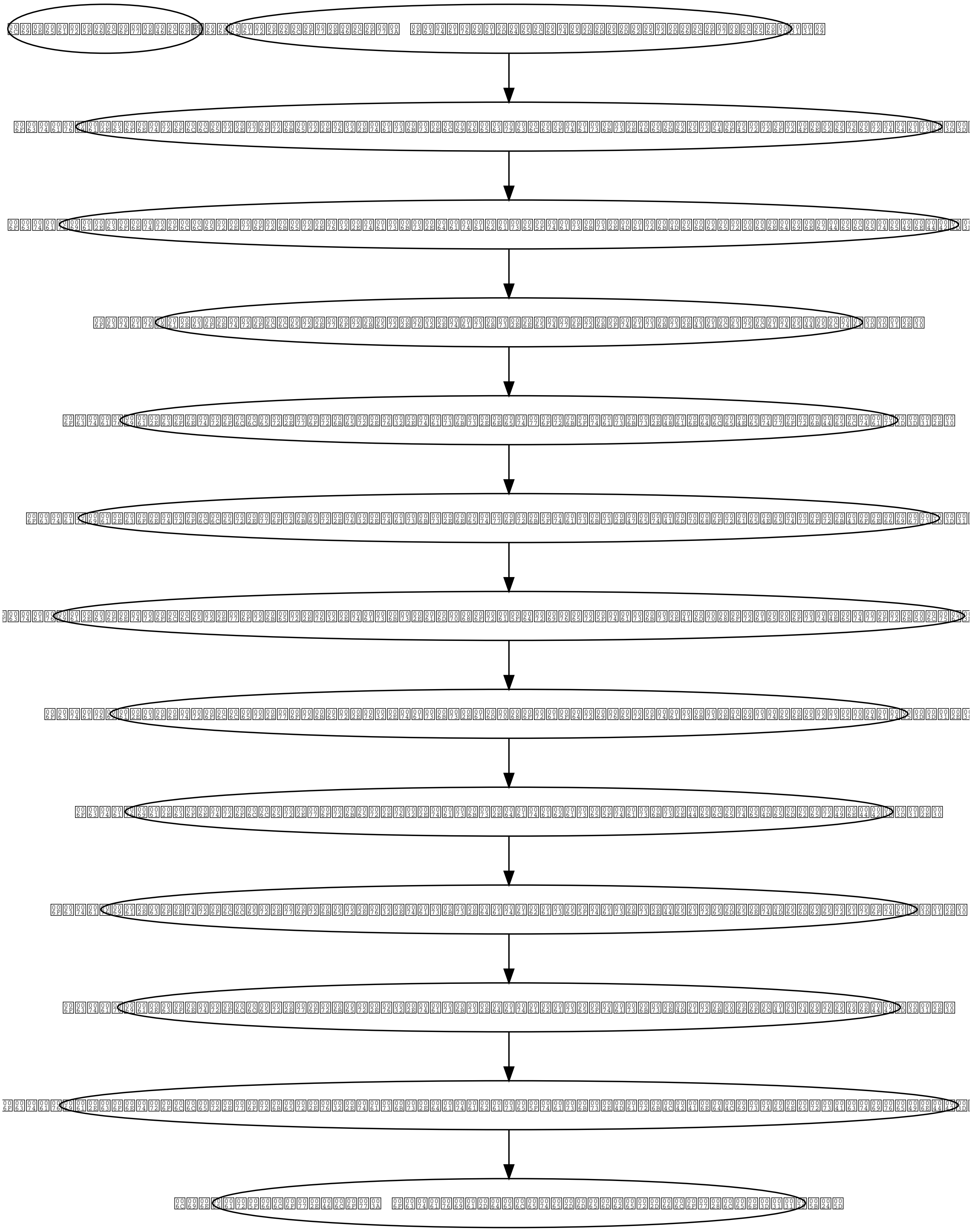
get_batch_update_members_flow



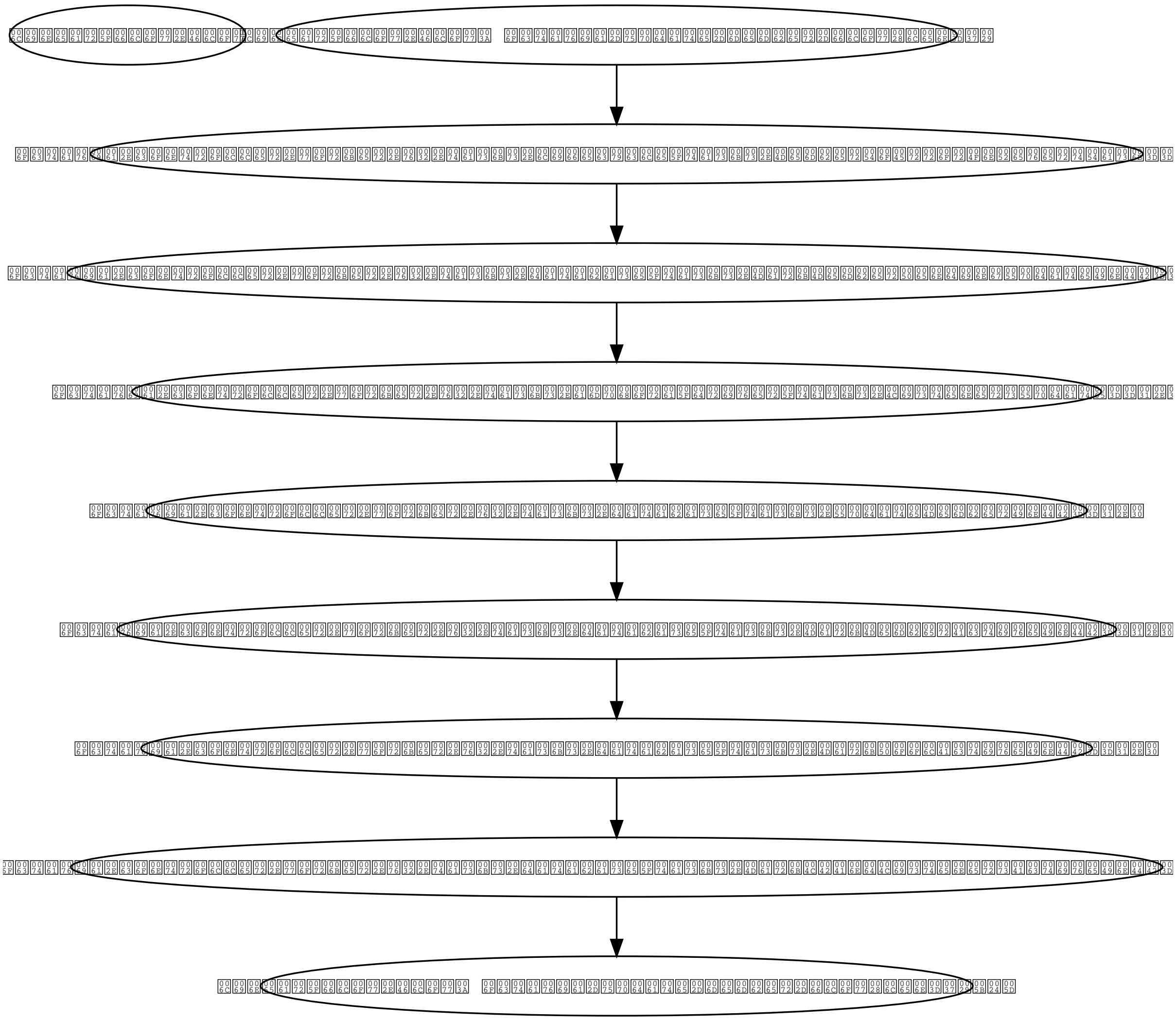
get_create_member_flow



get_delete_member_flow



get_update_member_flow

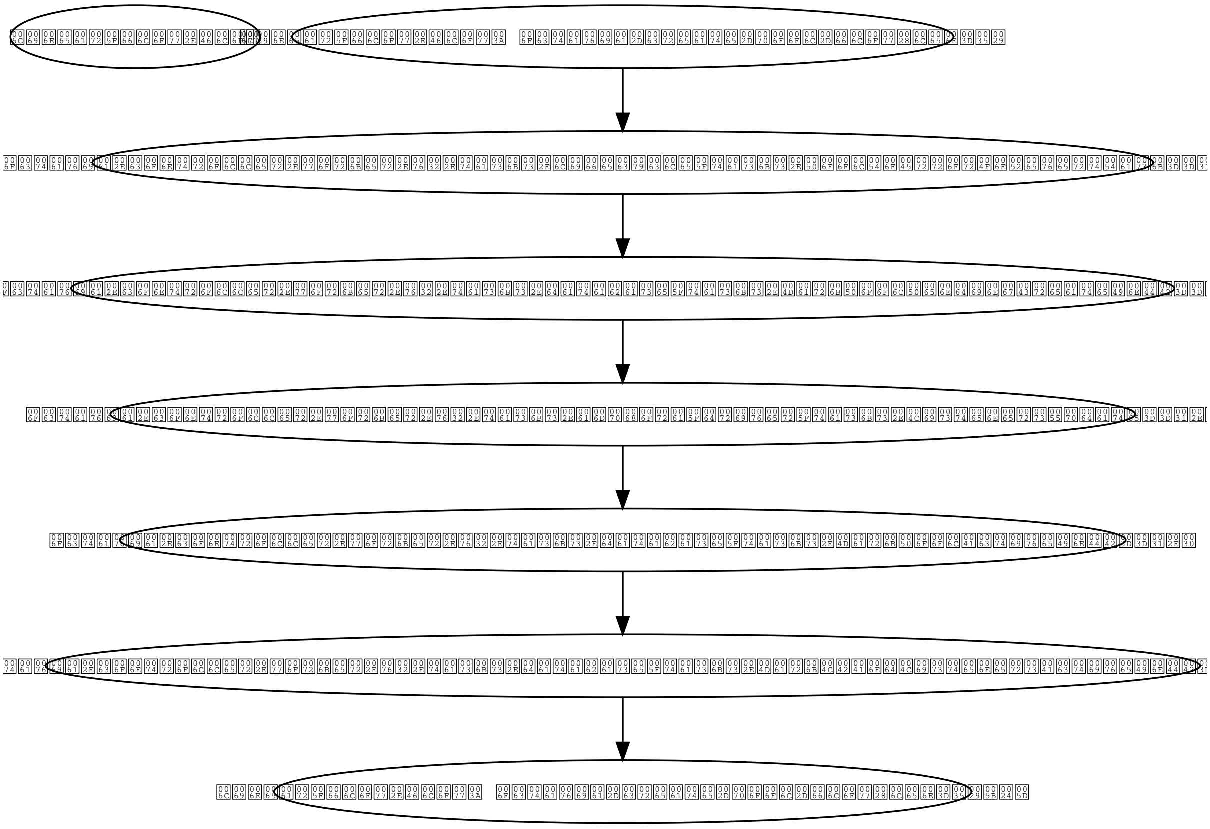


Pool Flows

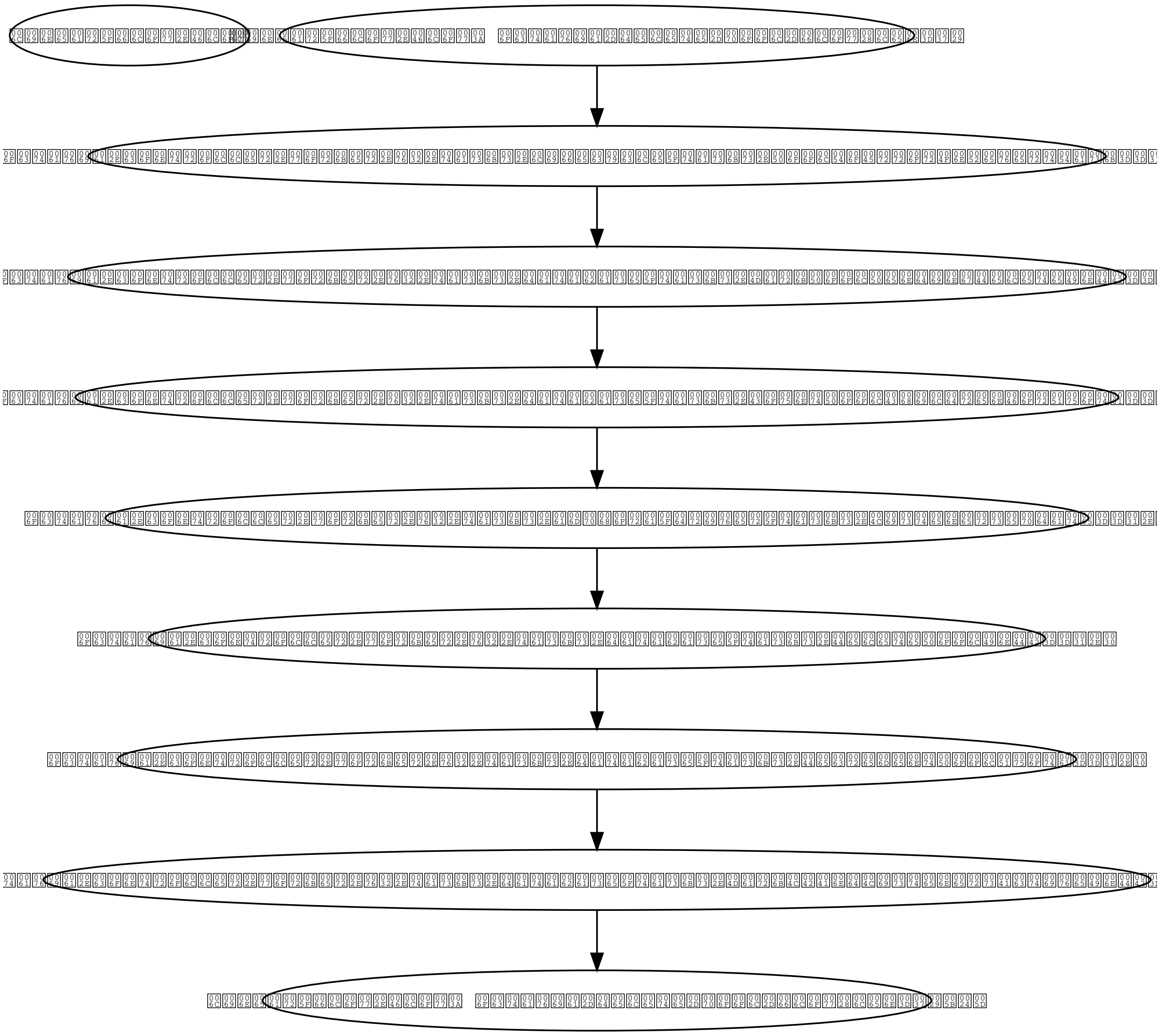
Contents

- *Pool Flows*
  - *get_create_pool_flow*
  - *get_delete_pool_flow*
  - *get_update_pool_flow*

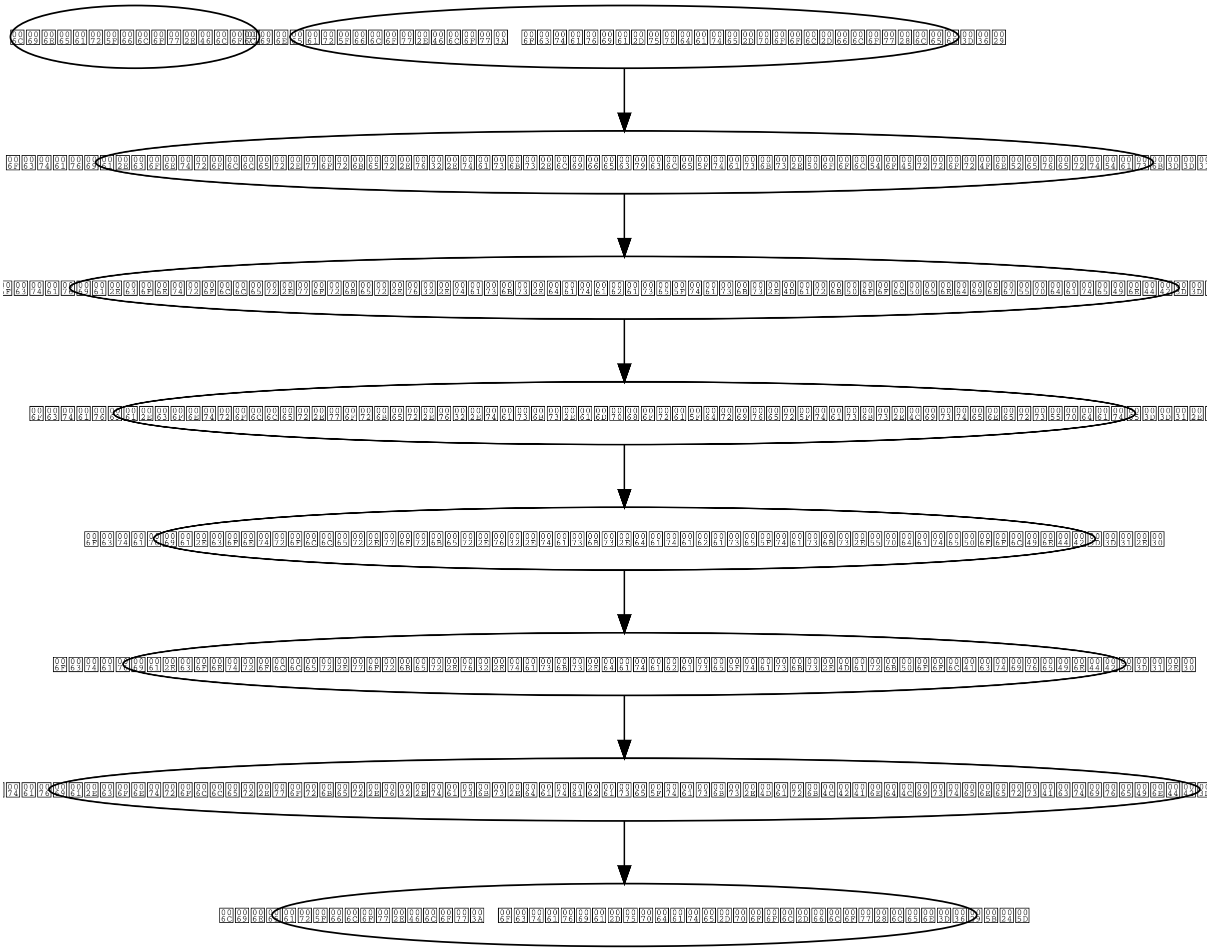
get_create_pool_flow



get_delete_pool_flow



get_update_pool_flow



4.2.5 Guru Meditation Reports

Octavia contains a mechanism whereby developers and system administrators can generate a report about the state of a running Octavia executable. This report is called a *Guru Meditation Report* (*GMR* for short).

Generating a GMR

A *GMR* can be generated by sending the *USR2* signal to any Octavia process with support (see below). The *GMR* will then be outputted as standard error for that particular process.

For example, suppose that `octavia-api` has process id `8675`, and was run with `2>/var/log/octavia/octavia-api-err.log`. Then, `kill -USR2 8675` will trigger the Guru Meditation report to be printed to `/var/log/octavia/octavia-api-err.log`.

Structure of a GMR

The *GMR* is designed to be extensible; any particular executable may add its own sections. However, the base *GMR* consists of several sections:

Package Shows information about the package to which this process belongs, including version information.

Threads Shows stack traces and thread ids for each of the threads within this process.

Green Threads Shows stack traces for each of the green threads within this process (green threads don't have thread ids).

Configuration Lists all the configuration options currently accessible via the CONF object for the current process.

Adding Support for GMRs to New Executables

Adding support for a *GMR* to a given executable is fairly easy.

First import the module:

```
from oslo_reports import guru_meditation_report as gmr
from octavia import version
```

Then, register any additional sections (optional):

```
TextGuruMeditation.register_section('Some Special Section',
                                    some_section_generator)
```

Finally (under main), before running the "main loop" of the executable (usually `service.server(server)` or something similar), register the *GMR* hook:

```
TextGuruMeditation.setup_autorun(version)
```

Extending the GMR

As mentioned above, additional sections can be added to the GMR for a particular executable. For more information, see the inline documentation under `oslo.reports`

4.3 Internal APIs

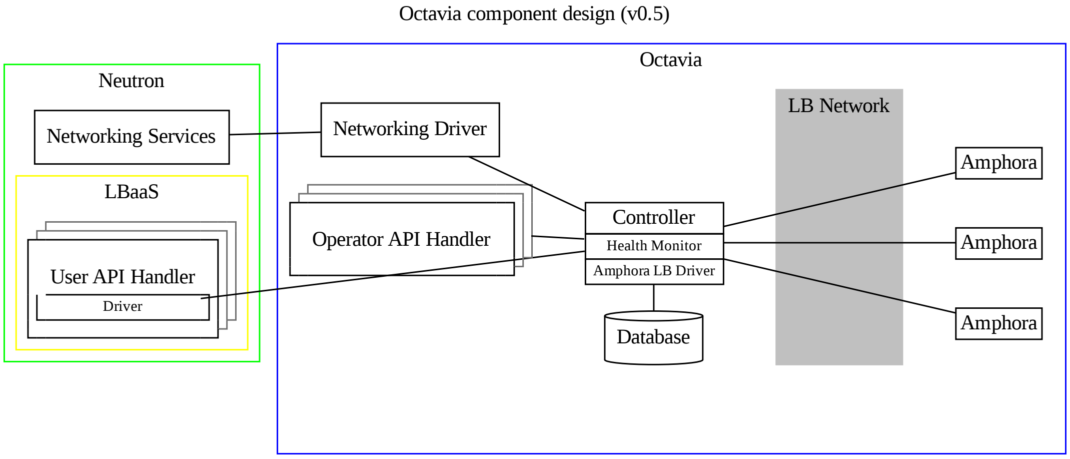
Note: The documents listed below are design documents and specifications created and approved at a previous point in time. The code base and current functionality may deviate from these original documents. Please see the Octavia documentation for the current feature details.

4.4 Design Documentation

4.4.1 Version 0.5 (liberty)

Octavia v0.5 Component Design

Please refer to the following diagram of the Octavia v0.5 components:


Octavia component design (v0.5)
Neutron
Networking Services
LBaaS
User API Handler
Driver
Octavia
Networking Driver
Operator API Handler
Controller
Health Monitor
Amphora LB Driver
Database
LB Network
Amphora
Amphora
Amphora

This milestone release of Octavia concentrates on making the service delivery scalable (though individual listeners are not horizontally scalable at this stage), getting API and other interfaces between major components correct, without worrying about making the command and control layer scalable.

Note that this design is not yet "operator grade" but is a good first step to achieving operator grade (which will occur with version 1 of Octavia).

LBaaS Components

The entities in this section describe components that are part of the Neutron LBaaS project, with which Octavia interfaces to deliver load balancing services.

USER API HANDLER

This is the front-end that users (and user GUIs or what have you) talk to manipulate load balancing services.

Notes:

- All implementation details are hidden from the user in this interface
- Performs a few simple sanity checks on user-supplied data, but otherwise looks to a driver provide more detail around whether what the user is asking for is possible on the driver's implementation.
- Any functionality that the user asks for that their back-end flavor / driver doesn't support will be met with an error when the user attempts to configure services this way. (There may be multiple kinds of errors: "incomplete configuration" would be non-fatal and allow DB objects to be created

/ altered. "incompatible configuration" would be fatal and disallow DB objects from being created / associations made.) Examples of this include: UDP protocol for a listener on a driver/flavor that uses only haproxy as its back-end.

- Drivers should also be able to return 'out of resources' or 'some other error occurred' errors (hopefully with helpful error messages).
- This interface is stateless, though drivers may keep state information in a database. In any case, this interface should be highly scalable.
- Talks some "intermediate driver interface" with the driver. This takes the form of python objects passed directly within the python code to the driver.

LBaaS / Octavia Crossover

The entities in this section are "glue" components which allow Octavia to interface with other services in the OpenStack environment. The idea here is that we want Octavia to be as loosely-coupled as possible with those services with which it must interact in order to keep these interfaces as clean as possible.

Initially, all the components in this section will be entirely under the purview of the Octavia project. Over time some of these components might be eliminated entirely, or reduced in scope as these third-party services evolve and increase in cleanly-consumable functionality.

DRIVER

This is the part of the load balancing service that actually interfaces between the (sanitized) user and operator configuration and the back-end load balancing appliances or other "service providing entity."

Notes:

- Configuration of the driver is handled via service profile definitions in association with the Neutron flavor framework. Specifically, a given flavor has service profiles associated with it, and service profiles which specify the Octavia driver will include meta-data (in the form of JSON configuration) which is used by the driver to define implementation specifics (for example, HA configuration and other details).
- Driver will be loaded by the daemon that does the user API and operator API. It is not, in and of itself, its own daemon, though a given vendor's back-end may contain its own daemons or other services that the driver interfaces with.
- It is thought that the driver front-end should be stateless in order to make it horizontally scalable and to preserves the statelessness of the user and operator API handlers. Note that the driver may interface with back-end components which need not be stateless.
- It is also possible for multiple instances of the driver will talk to the same amphora at the same time. Emphasis on the idempotency of the update algorithms used should help minimize the issues this can potentially cause.

NETWORK DRIVER

In order to keep Octavia's design more clean as a pure consumer of network services, yet still be able to develop Octavia at a time when it is impossible to provide the kind of load balancing services we need to provide without "going around" the existing Neutron API, we have decided to write a "network driver" component which does those dirty back-end configuration commands via an API we write, until these can become a standard part of Neutron. This component should be as loosely coupled with Octavia as Octavia will be with Neutron and present a standard interface to Octavia for accomplishing network configuration tasks (some of which will simply be a direct correlation with existing Neutron API commands).

Notes:

- This is a daemon or "unofficial extension", presumably living on a Neutron network node which should have "back door" access to all things Neutron and exposes an API that should only be used by Octavia.
- Exactly what API will be provided by this driver will be defined as we continue to build out the reference implementation for Octavia.
- Obviously, as we discover missing functionality in the Neutron API, we should work with the Neutron core devs to get these added to the API in a timely fashion: We want the Network driver to be as lightweight as possible.

Octavia Components

Everything from here down are entities that have to do with the Octavia driver and load balancing system. Other vendor drivers are unlikely to have the same components and internal structure. It is planned that Octavia will become the new reference implementation for LBaaS, though it of course doesn't need to be the only one. (In fact, a given operator should be able to use multiple vendors with potentially multiple drivers and multiple driver configurations through the Neutron Flavor framework.)

OPERATOR API HANDLER

This is exactly like the USER API HANDLER in function, except that implementation details are exposed to the operator, and certain admin-level features are exposed (ex. listing a given tenant's loadbalancers, & etc.)

It's also anticipated that the Operator API needs will vary enough from implementation to implementation that no single Operator API will be sufficient for the needs of all vendor implementations. (And operators will definitely have implementation-specific concerns.) Also, we anticipate that most vendors will already have an operator API or other interface which is controlled and configured outsite the purview of OpenStack in general. As such it makes sense for Octavia to have its own operator API / interface.

Notes:

- This interface is stateless. State should be managed by the controller, and stored in a highly available database.

CONTROLLER

This is the component providing all the command and control for the amphorae. On the front end, it takes its commands and controls from the LBaaS driver.

It should be noted that in later releases of Octavia, the controller functions will be split across several components. At this stage we are less concerned with how this internal communication will happen, and are most concerned with ensuring communication with amphorae, the amphora LB driver, and the Network driver are all made as perfect as possible.

Among the controller's responsibilities are:

- Sending configuration and certificate information to an amphora LB driver, which in the reference implementation will be generating configuration files for haproxy and PEM-formatted user certificates and sending these to individual amphorae. Configuration files will be generated from jinja templates kept in an template directory specific to the haproxy driver.
- Processing the configuration updates that need to be applied to individual amphorae, as sent by the amphora LB driver.
- Interfacing with network driver to plumb additional interfaces on the amphorae as necessary.
- Monitoring the health of all amphorae (via a driver interface).
- Receiving and routing certain kinds of notifications originating on the amphorae (ex. "member down")
- This is a stateful service, and should keep its state in a central, highly available database of some sort.
- Respecting colocation / apolocation requirements of loadbalancers as set forth by users.
- Receiving notifications, statistics data and other short, regular messages from amphorae and routing them to the appropriate entity.
- Responding to requests from amphorae for configuration data.
- Responding to requests from the user API or operator API handler driver for data about specific loadbalancers or sub-objects, their status, and statistics.
- Amphora lifecycle management, including interfacing with Nova and Neutron to spin up new amphorae as necessary and handle initial configuration and network plumbing for their LB network interface, and cleaning this up when an amphora is destroyed.
- Maintaining a pool of spare amphorae (ie. spawning new ones as necessary and deleting ones from the pool when we have too much inventory here.)
- Gracefully spinning down "dirty old amphorae"
- Loading and calling configured amphora drivers.

Notes:

- Almost all the intelligence around putting together and validating loadbalancer configurations will live here-- the Amphora API is meant to be as simple as possible so that minor feature improvements do not necessarily entail pushing out new amphorae across an entire installation.
- The size of the spare amphora pool should be determined by the flavor being offered.

- The controller also handles spinning up amphorae in the case of a true active/standby topology (ie. where the spares pool is effectively zero.) It should have enough intelligence to communicate to Nova that these amphorae should not be on the same physical host in this topology.
- It also handles spinning up new amphorae when one fails in the above topology.
- Since spinning up a new amphora is a task that can take a long time, the controller should spawn a job or child process which handles this highly asynchronous request.

AMPHORA LOAD BALANCER (LB) DRIVER

This is the abstraction layer that the controller talks to for communicating with the amphorae. Since we want to keep Octavia flexible enough so that certain components (like the amphora) can be replaced by third party products if the operator so desires, it's important to keep many of the implementation-specific details contained within driver layers. An amphora LB driver also gives the operator the ability to have different open-source amphorae with potentially different capabilities (accessed via different flavors) which can be handy for, for example, field-testing a new amphora image.

The reference implementation for the amphora LB driver will be for the amphora described below.

Responsibilities of the amphora LB driver include:

- Generating configuration files for haproxy and PEM-formatted user certificates and sending these to individual amphorae. Configuration files will be generated from jinja templates kept in an template directory specific to the haproxy driver.
- Handling all communication to and from amphorae.

LB NETWORK

This is the subnet that controllers will use to communicate with amphorae. This means that controllers must have connectivity (either layer 2 or routed) to this subnet in order to function, and vice versa. Since amphorae will be communicating on it, this means the network is not part of the "undercloud."

Notes:

- As certain sensitive data (TLS private keys, for example) will be transmitted over this communication infrastructure, all messages carrying a sensitive payload should be done via encrypted and authenticated means. Further, we recommend that messages to and from amphorae be signed regardless of the sensitivity of their content.

AMPHORAE

This is a Nova VM which actually provides the load balancing services as configured by the user. Responsibilities of these entities include:

- Actually accomplishing the load balancing services for user-configured loadbalancers using haproxy.
- Sending regular heartbeats (which should include some status information).
- Responding to specific requests from the controller for very basic loadbalancer or sub-object status data, including statistics.

- Doing common high workload, low intelligence tasks that we don't want to burden the controller with. (ex. Shipping listener logs to a swift data store, if configured.)
- Sending "edge" notifications (ie. status changes) to the controller when members go up and down, when listeners go up and down, etc.

Notes:

- Each amphora will generally need its own dedicated LB network IP address, both so that we don't accidentally bind to any IP:port the user wants to use for loadbalancing services, and so that an amphora that is not yet in use by any loadbalancer service can still communicate on the network and receive commands from its controller. Whether this IP address exists on the same subnet as the loadbalancer services it hosts is immaterial, so long as front-end and back-end interfaces can be plumbed after an amphora is launched.
- Since amphorae speak to controllers in a "trusted" way, it's important to ensure that users do not have command-line access to the amphorae. In other words, the amphorae should be a black box from the users' perspective.
- Amphorae will be powered using haproxy 1.5 initially. We may decide to use other software (especially for TLS termination) later on.
- The "glue scripts" which communicate with the controller should be as lightweight as possible: Intelligence about how to put together an haproxy config, for example, should not live on the amphora. Rather, the amphora should perform simple syntax checks, start / restart haproxy if the checks pass, and report success/failure of the haproxy restart.
- With few exceptions, most of the API commands the amphora will ever do should be safely handled synchronously (ie. nothing should take longer than a second or two to complete).
- Connection logs, and other things anticipated to generate a potential large amount of data should be communicated by the amphora directly to which ever service is going to consume that data. (for example, if logs are being shunted off to swift on a nightly basis, the amphora should handle this directly and not go through the controller.)

INTERNAL HEALTH MONITORS

There are actually a few of these, all of which need to be driven by some daemon(s) which periodically check that heartbeats from monitored entities are both current and showing "good" status, if applicable. Specifically:

- Controllers need to be able to monitor the availability and overall health of amphorae they control. For active amphorae, this check should happen pretty quickly: About once every 5 seconds. For spare amphorae, the check can happen much more infrequently (say, once per minute).

The idea here is that internal health monitors will monitor a periodic heartbeat coming from the amphorae, and take appropriate action (assuming these are down) if they fail to check in with a heartbeat frequently enough. This means that internal health monitors need to take the form of a daemon which is constantly checking for and processing heartbeat requests (and updating controller or amphorae statuses, and triggering other events as appropriate).

Some notes on Controller <-> Amphorae communications

In order to keep things as scalable as possible, the thought was that short, periodic and arguably less vital messages being emitted by the amphora and associated controller would be done via HMAC-signed UDP, and more vital, more sensitive, and potentially longer transactional messages would be handled via a RESTful API on the controller, accessed via bi-directionally authenticated HTTPS.

Specifically, we should expect the following to happen over UDP: * heartbeats from the amphora VM to the controller

- stats data from the amphora to the controller
- "edge" alert notifications (change in status) from the amphora to the controller
- Notification of pending tasks in queue from controller to amphora

And the following would happen over TCP: * haproxy / tls certificate configuration changes

Supported Amphora Virtual Appliance Topologies

Initially, I propose we support two topologies with version 0.5 of Octavia:

Option 1: "Single active node + spares pool"

- This is similar to what HP is doing right now with Libra: Each amphora is stand-alone with a frequent health-check monitor in place and upon failure, an already-spun-up amphora is moved from the spares pool and configured to take the old one’s place. This allows for acceptable recovery times on amphora failure while still remaining efficient, as far as VM resource utilization is concerned.

Option 2: "True Active / Standby"

- This is similar to what Blue Box is doing right now where amphorae are deployed in pairs and use corosync / pacemaker to monitor each other’s health and automatically take over (usually in less than 5 seconds) if the "active" node fails. This provides for the fastest possible recovery time on hardware failure, but is much less efficient, as far as VM resource utilization is concerned.
- In this topology a floating IP address (different from a Neutron floating IP!) is used to determine which amphora is the "active" one at any given time.
- In this topology, both amphorae need to be colocated on the same subnet. As such a "spares pool" doesn’t make sense for this type of layout, unless all spares are on the same management network with the active nodes.

We considered also supporting "Single node" topology, but this turns out to be the same thing as option 1 above with a spares pool size of zero.

Supported Network Topologies

This is actually where things get tricky, as far as amphora plumbing is concerned. And it only grows trickier when we consider that front-end connectivity (ie. to the 'loadbalancer' vip_address) and back-end connectivity (ie. to members of a loadbalancing pool) can be handled in different ways. Having said this, we can break things down into LB network, front-end and back-end topology to discuss the various possible permutations here.

LB Network

Each amphora needs to have a connection to a LB network. And each controller needs to have access to this management network (this could be layer-2 or routed connectivity). Command and control will happen via the amphorae's LB network IP.

Front-end topologies

There are generally two ways to handle the amphorae's connection to the front-end IP address (this is the vip_address of the loadbalancer object):

Option 1: Layer-2 connectivity

The amphora can have layer-2 connectivity to the neutron network which is host to the subnet on which the loadbalancer vip_address resides. In this scenario, the amphora would need to send ARP responses to requests for the vip_address, and therefore amphorae need to have interfaces plumbed on said vip_address subnets which participate in ARP.

Note that this is somewhat problematic for active / standby virtual appliance topologies because the vip_address for a given load balancer effectively becomes a highly-available IP address (a true floating VIP), which means on service failover from active to standby, the active amphora needs to relinquish all the vip_addresses it has, and the standby needs to take them over *and* start up haproxy services. This is OK if a given amphora only has a few load balancers, but can lead to several minutes' down-time during a graceful failover if there are a dozen or more load balancers on the active/standby amphora pair. It's also more risky: The standby node might not be able to start up all the haproxy services during such a failover. What's more, most types of VRRP-like services which handle floating IPs require amphorae to have an additional IP address on the subnet housing the floating vip_address in order for the standby amphora to monitor the active amphora.

Also note that in this topology, amphorae need an additional virtual network interface plumbed when new front-end loadbalancer vip_addresses are assigned to them which exist on subnets to which they don't already have access.

Option 2: Routed (layer-3) connectivity

In this layout, static routes are injected into the routing infrastructure (Neutron) which essentially allow traffic destined for any given loadbalancer vip_address to be routed to an IP address which lives on the amphora. (I would recommend this be something other than the LB network IP.) In this topology, it's actually important that the loadbalancer vip_address does *not* exist in any subnet with potential front-end clients because in order for traffic to reach the loadbalancer, it must pass through the routing infrastructure (and in this case, front-end clients would attempt layer-2 connectivity to the vip_address).

This topology also works much better for active/standby configurations, because both the active and standby amphorae can bind to the vip_addresses of all their assigned loadbalancer objects on a dummy, non-ARPing interface, both can be running all haproxy services at the same time, and keep the standby server processes from interfering with active loadbalancer traffic through the use of fencing scripts on the amphorae. Static routing is accomplished to a highly available floating "routing IP" (using some VRRP-like service for just this IP) which becomes the trigger for the fencing scripts on the amphora. In this scenario, fail-overs are both much more reliable, and can be accomplished in usually < 5 seconds.

Further, in this topology, amphorae do not need any additional virtual interfaces plumbed when new front-end loadbalancer vip_addresses are assigned to them.

Back-end topologies

There are also two ways that amphorae can potentially talk to back-end member IP addresses. Unlike the front-end topologies (where option 1 and option 2 are basically mutually exclusive, if not practically exclusive) both of these types of connectivity can be used on a single amphora, and indeed, within a single loadbalancer configuration.

Option 1: Layer-2 connectivity

This is layer-2 connectivity to back-end members, and is implied when a member object has a subnet_id assigned to it. In this case, the existence of the subnet_id implies amphorae need to have layer-2 connectivity to that subnet, which means they need to have a virtual interface plumbed to it, as well as an IP address on the subnet. This type of connectivity is useful for "secure" back-end subnets that exist behind a NATing firewall where PAT is not in use on the firewall. (In this way it effectively bypasses the firewall.) We anticipate this will be the most common form of back-end connectivity in use by most OpenStack users.

Option 2: Routed (layer-3) connectivity

This is routed connectivity to back-end members. This is implied when a member object does not have a subnet_id specified. In this topology, it is assumed that member ip_addresses are reachable through standard neutron routing, and therefore connections to them can be initiated from the amphora's default gateway. No new virtual interfaces need to be plumbed for this type of connectivity to members.

4.5 Project Specifications

4.5.1 Version 0.5 (liberty)

Amphora Driver Interface

https://blueprints.launchpad.net/octavia/+spec/amphora-driver-interface

This blueprint describes how a driver will interface with the controller. It will describe the base class and other classes required. It will not describe the REST interface needed to talk to an amphora nor how health information or statistics are gathered from the amphora.

Problem description

The controller needs to talk through a driver to the amphora to allow for custom APIs and custom rendering of configuration data for different amphora implementations.

The controller will heavily utilize taskflow [2] to accomplish its goals so it is highly encouraged for drivers to use taskflow to organize their work, too.

Proposed change

Establish a base class to model the desire functionality:

```
class AmphoraLoadBalancerDriver(object):

    def update(self, listener, vip):
        """updates the amphora with a new configuration

        for the listener on the vip.
        """
        raise NotImplementedError

    def stop(self, listener, vip):
        """stops the listener on the vip."""
        return None

    def start(self, listener, vip):
        """starts the listener on the vip."""
        return None

    def delete(self, listener, vip):
        """deletes the listener on the vip."""
        raise NotImplementedError

    def get_info(self, amphora):
        """Get detailed information about an amphora

        returns information about the amphora, e.g. {"Rest Interface":
        "1.0", "Amphorae": "1.0", "packages":{"ha proxy":"1.5"},
        "network-interfaces": {"eth0":{"ip":...}} some information might
        come from querying the amphora
        """
        raise NotImplementedError

    def get_diagnostics(self, amphora):
        """OPTIONAL - Run diagnostics

        run some expensive self tests to determine if the amphora and the
        lbs are healthy the idea is that those tests are triggered more
        infrequent than the heartbeat
        """
```

(continues on next page)

(continued from previous page)

```
        raise NotImplementedError

    def finalize_amphora(self, amphora):
        """OPTIONAL - called once an amphora has been build but before

        any listeners are configured. This is a hook for drivers who need
        to do additional work before am amphora becomes ready to accept
        listeners. Please keep in mind that amphora might be kept in am
        offline pool after this call.
        """
        pass

    def post_network_plug(self, amphora, port):
        """OPTIONAL - called after adding a compute instance to a network.

        This will perform any necessary actions to allow for connectivity
        for that network on that instance.

        port is an instance of octavia.network.data_models.Port.  It
        contains information about the port, subnet, and network that
        was just plugged.
        """

    def post_vip_plug(self, load_balancer, amphorae_network_config):
        """OPTIONAL - called after plug_vip method of the network driver.

        This is to do any additional work needed on the amphorae to plug
        the vip, such as bring up interfaces.

        amphorae_network_config is a dictionary of objects that include
        network specific information about each amphora's connections.
        """

    def start_health_check(self, health_mixin):
        """start check health

        :param health_mixin: health mixin object
        :type amphora: object

        Start listener process and  calls HealthMixin to update
        databases information.
        """
        pass

    def stop_health_check(self):
        """stop check health

        Stop listener process and  calls HealthMixin to update
        databases information.
```

(continues on next page)

(continued from previous page)

```
        """
        pass
```

The referenced listener is a listener object and vip a vip as described in our model. The model is detached from the DB so the driver can't write to the DB. Because our initial goal is to render a whole config no special methods for adding nodes, health monitors, etc. are supported at this juncture. This might be added in later versions.

No method for obtaining logs has been added. This will be done in a future blueprint.

Exception Model

The driver is expected to raise the following well defined exceptions

- NotImplementedError - this functionality is not implemented/not supported
- **AmphoraDriverError - a super class for all other exceptions and the catch** all if no specific exception can be found
  - NotFoundError - this amphora couldn't be found/ was deleted by nova
  - InfoException - gathering information about this amphora failed
  - NetworkConfigException - gathering network information failed
  - UnauthorizedException - the driver can't access the amphora
  - TimeOutException - contacting the amphora timed out
  - UnavailableException - the amphora is temporary unavailable
  - SuspendFaied - this load balancer couldn't be suspended
  - EnableFailed - this load balancer couldn't be enabled
  - DeleteFailed - this load balancer couldn't be deleted
  - ProvisioningErrors - those are errors which happen during provisioning
    - ListenerProvisioningError - could not provision Listener
    - LoadBalancerProvisoningError - could not provision LoadBalancer
    - HealthMonitorProvisioningError - could not provision HealthMonitor
    - NodeProvisioningError - could not provision Node

Health and Stat Mixin

It has been suggested to gather health and statistic information via UDP packets emitted from the amphora. This requires each driver to spin up a thread to listen on a UDP port and then hand the information to the controller as a mixin to make sense of it.

Here is the mixin definition:

```
class HealthMixIn(object):
    def update_health(health):
        #map: {"amphora-status":HEALTHY, loadbalancers: {"loadbalancer-id": {
↪"loadbalancer-status": HEALTHY,
        # "listeners":{"listener-id":{"listener-status":HEALTHY, "nodes":{
↪"node-id":HEALTHY, ...}}, ...}, ...}}
        # only items whose health has changed need to be submitted
        # awesome update code
        pass

class StatsMixIn(object):
    def update_stats(stats):
        #uses map {"loadbalancer-id":{"listener-id": {"bytes-in": 123, "bytes_
↪out":123, "active_connections":123,
        # "total_connections", 123}, ...}
        # elements are named to keep it extensible for future versions
        #awesome update code and code to send to ceilometer
        pass
```

Things a good driver should do:

- Non blocking IO - throw an appropriate exception instead to wait forever; use timeouts on sockets
- We might employ a circuit breaker to insulate driver problems from controller problems [1]
- Use appropriate logging
- Use the preferred threading model

This will be demonstrated in the Noop-driver code.

Alternatives

Require all amphora to implement a common REST interface and use that as the integration point.

Data model impact

None

REST API impact

None

Security impact

None

Notifications impact

None - since initial version

Other end user impact

None

Performance Impact

Minimal

Other deployer impact

Deployers need to make sure to bundle the compatible versions of amphora, driver, controller --

Developer impact

Need to write towards this clean interface.

Implementation

Assignee(s)

German Eichberger

Work Items

- Write abstract interface
- Write Noop driver
- Write tests

Dependencies

None

Testing

- Unit tests with tox and Noop-Driver
- tempest tests with Noop-Driver

Documentation Impact

None - we won't document the interface for 0.5. If that changes we need to write an interface documentation so 3rd party drivers know what we expect.

References

[1] https://martinfowler.com/bliki/CircuitBreaker.html [2] https://docs.openstack.org/taskflow/latest/

Compute Driver Interface

https://blueprints.launchpad.net/octavia/+spec/compute-driver-interface

This blueprint describes how a driver will interface with Nova to manage the creation and deletion of amphora instances. It will describe the base class and other classes required to create, delete, manage the execution state, and query the status of amphorae.

Problem description

The controller needs to be able to create, delete, and monitor the status of amphora instances. The amphorae may be virtual machines, containers, bare-metal servers, or dedicated hardware load balancers. This interface should hide the implementation details of the amphorae from the caller to the maximum extent possible.

This interface will provide means to specify:

- type (VM, Container, bare metal)
- flavor (provides means to specify memory and storage capacity)
- what else?

Proposed change

Establish an abstract base class to model the desired functionality:

```
class AmphoraComputeDriver(object):

    def build(self, amphora_type = VM, amphora_flavor = None,
              image_id = None, keys = None, sec_groups = None,
              network_ids = None,config_drive_files = None,user_data=None):

        """ build a new amphora.

        :param amphora_type: The type of amphora to create.  For
        version 0.5, only VM is supported.  In the future this
        may support Container, BareMetal, and HWLoadBalancer.
        :param amphora_flavor: Optionally specify a flavor.  The
        interpretation of this parameter will depend upon the
        amphora type and may not be applicable to all types.
        :param image_id: ID of the base image for a VM amphora
        :param keys: Optionally specify a list of ssh public keys
        :param sec_groups: Optionally specify list of security
        groups
        :param network_ids: A list of network_ids to attach to
        the amphora
        :config_drive_files: A dict of files to overwrite on
        the server upon boot. Keys are file names (i.e. /etc/passwd)
        and values are the file contents (either as a string or as
        a file-like object). A maximum of five entries is allowed,
        and each file must be 10k or less.
        :param user_data: user data to pass to be exposed by the
        metadata server this can be a file type object as well or
        a string

        :returns: The id of the new instance.

        """

        raise NotImplementedError

    def delete(self, amphora_id):
        """  delete the specified amphora
        """

        raise NotImplementedError

    def status(self, amphora_id):

        """ Check whether the specified amphora is up

        :param amphora_id: the ID of the desired amphora
```

(continues on next page)

(continued from previous page)

```
        :returns: the nova response from the amphora
        """
        raise NotImplementedError

    def get_amphora(self, amphora_name = None, amphora_id = None):
        """ Try to find an amphora given its name or id

        :param amphora_name: the name of the desired amphora
        :param amphora_id: the id of the desired amphora
        :returns: the amphora object
        """
        raise NotImplementedError
```

Exception Model

The driver is expected to raise the following well defined exceptions:

- NotImplementedError - this functionality is not implemented/not supported
- **AmphoraComputeError - a super class for all other exceptions and the catch** all if no specific exception can be found
  - AmphoraBuildError - An amphora of the specified type could not be built
  - DeleteFailed - this amphora couldn't be deleted
- InstanceNotFoundError - an instance matching the desired criteria could not be found
- NotSuspendedError - resume() attempted on an instance that was not suspended

Things a good driver should do:

- Non blocking operations - If an operation will take a long time to execute, perform it asynchronously. The definition of "a long time" is open to interpretation, but a common UX guideline is 200 ms
- We might employ a circuit breaker to insulate driver problems from controller problems [1]
- Use appropriate logging
- Use the preferred threading model

This will be demonstrated in the Noop-driver code.

Alternatives

Data model impact

None

REST API impact

None

Security impact

None

Notifications impact

None - since initial version

Other end user impact

None

Performance Impact

Minimal

Other deployer impact

Deployers need to make sure to bundle the compatible versions of amphora, driver, controller --

Developer impact

Need to write towards this clean interface.

Implementation

Assignee(s)

Al Miller

Work Items

- Write abstract interface
- Write Noop driver
- Write tests

Dependencies

None

Testing

- Unit tests with tox and Noop-Driver
- tempest tests with Noop-Driver

Documentation Impact

None - this is an internal interface and need not be externally documented.

References

[1] http://martinfowler.com/bliki/CircuitBreaker.html

Octavia Base Image

Launchpad blueprint:

https://blueprints.launchpad.net/octavia/+spec/base-image

Octavia is an operator-grade reference implementation for Load Balancing as a Service (LBaaS) for OpenStack. The component of Octavia that does the load balancing is known as amphora. Amphora may be a virtual machine, may be a container, or may run on bare metal. Creating images for bare metal amphora installs is outside the scope of this 0.5 specification but may be added in a future release.

Amphora will need a base image that can be deployed by Octavia to provide load balancing.

Problem description

Octavia needs a method for generating base images to be deployed as load balancing entities.

Proposed change

Leverage the OpenStack diskimage-builder project [1] tools to provide a script that builds qcow2 images or a tar file suitable for use in creating containers. This script will be modeled after the OpenStack Sahara [2] project's diskimage-create.sh script.

This script and associated elements will build Amphora images. Initial support with be with an Ubuntu OS and HAProxy. The script will be able to use Fedora or CentOS as a base OS but these will not initially be tested or supported. As the project progresses and/or the diskimage-builder project adds support for additional base OS options they may become available for Amphora images. This does not mean that they are necessarily supported or tested.

The script will use environment variables to customize the build beyond the Octavia project defaults, such as adding elements.

The initial supported and tested image will be created using the diskimage-create.sh defaults (no command line parameters or environment variables set). As the project progresses we may add additional supported configurations.

Command syntax:

```
$ diskimage-create.sh
        [-a i386 | amd64 | armhf ]
        [-b haproxy ]
        [-c ~/.cache/image-create | <cache directory> ]
        [-h]
        [-i ubuntu | fedora | centos ]
        [-o amphora-x64-haproxy | <filename> ]
        [-r <root password> ]
        [-s 5 | <size in GB> ]
        [-t qcow2 | tar ]
        [-w <working directory> ]
    '-a' is the architecture type for the image (default: amd64)
    '-b' is the backend type (default: haproxy)
    '-c' is the path to the cache directory (default: ~/.cache/image-create)
    '-h' display help message
    '-i' is the base OS (default: ubuntu)
    '-o' is the output image file name
    '-r' enable the root account in the generated image (default: disabled)
    '-s' is the image size to produce in gigabytes (default: 5)
    '-t' is the image type (default: qcow2)
    '-w' working directory for image building (default: .)
```

Environment variables supported by the script:

DIB_DISTRIBUTION_MIRROR - URL to a mirror for the base OS selected (-i).
DIB_REPO_PATH - Path to the diskimage-builder repository (default: ../../diskimage-builder)
ELEMENTS_REPO_PATH - Path to the /tripleo-image-elements repository (default:

../../tripleo-image-elements)

DIB_ELEMENTS - Override the elements used to build the image

DIB_LOCAL_ELEMENTS - Elements to add to the build (requires DIB_LOCAL_ELEMENTS_PATH be specified)

DIB_LOCAL_ELEMENTS_PATH - Path to the local elements directory

Container support

The Docker command line required to import a tar file created with this script is [3]:

```
$ docker import - image:amphora-x64-haproxy < amphora-x64-haproxy.tar
```

Alternatives

Deployers can manually create an image or container, but they would need to make sure the required components are included.

Data model impact

None

REST API impact

None

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

This script will make creating an Octavia Amphora image or container simple.

Developer impact

None

Implementation

Assignee(s)

Michael Johnson <johnsom>

Work Items

1. Write diskimage-create.sh script based on Sahara project's script.
2. Identify the list of packages required for Octavia Amphora.
3. Create required elements not provided by the diskimage-builder project.
4. Create unit tests

Dependencies

This script will depend on the OpenStack diskimage-builder project.

Testing

Initial testing will be completed using the default settings for the diskimage-create.sh tool.

- **Unit tests with tox**
    - Validate that the image is the correct size and mounts via loopback
    - Check that a valid kernel is installed
    - Check that HAProxy and all required packages are installed
- tempest tests

Documentation Impact

References

[1] https://github.com/openstack/diskimage-builder
[2] https://github.com/openstack/sahara-image-elements
[3] https://github.com/openstack/diskimage-builder/blob/master/docs/docker.md

Octavia v0.5 master component design document

Problem description

We need to define the various components that will make up Octavia v0.5.

Proposed change

This is the first functional release of Octavia, incorporating a scalable service delivery layer, but not yet concerned with a scalable command and control layer.

See doc/source/design/version0.5 for a detailed description of the v0.5 component design.

Alternatives

We're open to suggestions, but note that later designs already discussed on the mailing list will incorporate several features of this design.

Data model impact

Octavia 0.5 introduces the main data model which will also be used in subsequent releases.

REST API impact

None

Security impact

The only sensitive data used in Octavia 0.5 are the TLS private keys used with TERMINATED_HTTPS functionality. However, the back-end storage aspect of these secrets will be handled by Barbican.

Octavia amphorae will also need to keep copies of these secrets locally in order to facilitate seamless service restarts. These local stores should be made on a memory filesystem.

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

Operator API and UI may need to be changed as a result of this specification.

Developer impact

None beyond implementing the spec. :)

Implementation

Assignee(s)

Lots of us will be working on this!

Work Items

Again, lots of things to be done here.

Dependencies

Barbican

Testing

A lot of new tests will need to be written to test the separate components, their interfaces, and likely failure scenarios.

Documentation Impact

This specification largely defines the documentation of the component design.

Component design is becoming a part of the project standard documentation.

References

Mailing list discussion of similar designs earlier this year

Octavia Controller

Launchpad blueprint:

https://blueprints.launchpad.net/octavia/+spec/controller

Octavia is an operator-grade reference implementation for Load Balancing as a Service (LBaaS) for OpenStack. The component of Octavia that does the load balancing is known as Amphora.

The component of Octavia that provides command and control of the Amphora is the Octavia controller.

Problem description

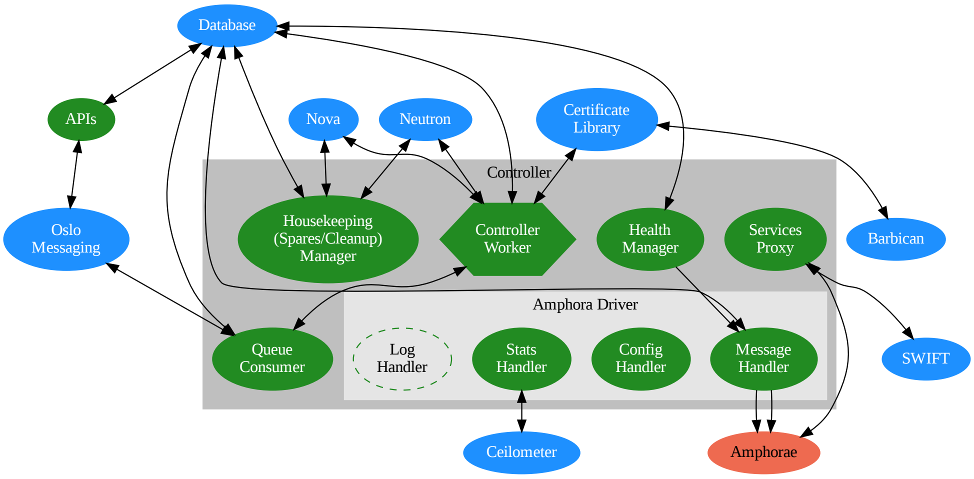
Octavia requires a controller component that provides the following capabilities:

- Processing Amphora configuration updates and making them available to the Amphora driver
- Providing certificate information to the Amphora driver
- Deploying Amphora instances
- Managing the Amphora spares pool
- Cleaning up Amphora instances that are no longer needed
- Monitoring the health of Amphora instances
- Processing alerts and messages from the Amphora (example "member down")
- Respecting colocation / apolocation / flavor requirements of the Amphora
- Processing statistical data from the Amphora including communicating with metering services, such as Ceilometer (https://blueprints.launchpad.net/ceilometer/+spec/ceilometer-meter-lbaas)
- Responding to API requests sent by the API processes
- Proxy Amphora data to other OpenStack services such as Swift for log file archival

Proposed change

The Octavia controller will consist of the following components:

- Amphora Driver
- Queue Consumer
- Certificate Library
- Compute Driver
- Controller Worker
- Health Manager
- Housekeeping Manager
- Network Driver
- Services Proxy


Database
APIs
Nova
Neutron
Certificate Library
Controller
Oslo Messaging
Housekeeping (Spares/Cleanup) Manager
Controller Worker
Health Manager
Services Proxy
Barbican
Amphora Driver
Queue Consumer
Log Handler
Stats Handler
Config Handler
Message Handler
SWIFT
Ceilometer
Amphorae

The manager and proxy components should be implemented as independent processes to provide a level of autonomy to these controller functions.

The highly available database will provide the persistent "brain" for the Octavia controller. Octavia controller processes will share state and information about the Amphora, load balancers, and listeners via the database. It is expected that the Octavia controller and Amphora driver will directly interact with the database but the Amphorae will never directly access the database.

By using a highly available database, Octavia controllers themselves do not directly keep any stateful information on Amphorae. Because of this, Amphorae are not assigned to any specific controller. Any controller is able to service monitoring, heartbeat, API, and other requests coming to or from Amphorae.

Amphora Driver

The Amphora driver abstracts the backend implementation of an Amphora. The controller will interact with Amphora via the Amphora driver. This interface is defined in the amphora-driver-interface specification.

Queue Consumer

The Queue Consumer is event driven and tasked with servicing requests from the API components via an Oslo messaging. It is also the primary lifecycle management component for Amphora.

To service requests the Queue Consumer will spawn a Controller Worker process. Spawning a separate process makes sure that the Queue Consumer can continue to service API requests while the longer running deployment process is progressing.

Messages received via Oslo messaging will include the load balancer ID, requested action, and configuration update data. Passing the configuration update data via Oslo messaging allows the deploy worker to rollback to a "last known good" configuration should there be a problem with the configuration update. The spawned worker will use this information to access the Octavia database to gather any additional details that may be required to complete the requested action.

Compute Driver

The Compute Driver abstracts the implementation of instantiating the virtual machine, container, appliance, or device that the Amphora will run in.

Controller Worker

The Controller Worker is spawned from the Queue Consumer or the Health Manager. It interfaces with the compute driver (in some deployment scenarios), network driver, and Amphora driver to activate Amphora instances, load balancers, and listeners.

When a request for a new instance or failover is received the Controller Worker will have responsibility for connecting the appropriate networking ports to the Amphora via the network driver and triggering a configuration push via the Amphora driver. This will include validating that the targeted Amphora has the required networks plumbed to the Amphora.

The Amphora configured by the Controller Worker may be an existing Amphora instance, a new Amphora from the spares pool, or a newly created Amphora. This determination will be made based on the apolocation requirements of the load balancer, the load balancer count on the existing Amphora, and the availability of ready spare Amphora in the spares pool.

The Controller Worker will be responsible for passing in the required metadata via config drive when deploying an Amphora. This metadata will include: a list of controller IP addresses, controller certificate authority certificate, and the Amphora certificate and key file.

The main flow of the Controller Worker is described in the amphora-lifecycle-management specification as the Activate Amphora sequence.

Certificate Library

The Certificate Library provides an abstraction for workers to access security data stored in OpenStack Barbican from the Amphora Driver. It will provide a short term (1 minute) cache of the security contents to facilitate the efficient startup of a large number of listeners sharing security content.

Health Manager

The Health Manager is tasked with checking for missing or unhealthy Amphora stored in the highly available database. The amphora-lifecycle-management specification details the health monitoring sequence.

The health monitor will have a separate thread that checks these timestamps on a configurable interval to see if the Amphora has not provided a heartbeat in the required amount of time which is another configurable setting. Should a Amphora fail to report a heartbeat in the configured interval the Health Manager will initiate a failover of the Amphora by spawning a deploy worker and will update the status of the listener in the database.

The Health Manager will have to be aware of the load balancer associated with the failed listener to decide if it needs to fail over additional listeners to migrate the failed listener to a new Amphora.

Housekeeping Manager

The Housekeeping Manager will manage the spare Amphora pool and the teardown of Amphora that are no longer needed. On a configurable interval the Housekeeping Manager will check the Octavia database to identify the required cleanup and maintenance actions. The amphora-lifecycle-management specification details the Create, Spare, and Delete Amphora sequences the Housekeeping Manager will follow.

The operator can specify a number of Amphora instances to be held in a spares pool. Building Amphora instances can take a long time so the Housekeeping Manager will spawn threads to manage the number of Amphorae in the spares pool.

The Housekeeping Manager will interface with the compute driver, network driver, and the Certificate Manager to accomplish the create and delete actions.

Network Driver

The Network Driver abstracts the implementation of connecting an Amphora to the required networks.

Services Proxy

The Services Proxy enables Amphora to reach other cloud services directly over the Load Balancer Network where the controller may need to provide authentication tokens on behalf of the Amphora, such as when archiving load balancer traffic logs into customer swift containers.

Alternatives

Data model impact

REST API impact

Security impact

Notifications impact

Other end user impact

Performance Impact

Other deployer impact

Developer impact

Implementation

Assignee(s)

Michael Johnson <johnsom>

Work Items

Dependencies

Testing

Documentation Impact

References

Amphora lifecycle management: https://review.opendev.org/#/c/130424/
LBaaS metering:
https://blueprints.launchpad.net/ceilometer/+spec/ceilometer-meter-lbaas

Controller Worker (deploy-worker)

Launchpad blueprint:

https://blueprints.launchpad.net/octavia/+spec/controller-worker

Octavia is an operator-grade reference implementation for Load Balancing as a Service (LBaaS) for OpenStack. The component of Octavia that does the load balancing is known as Amphora.

The component of Octavia that provides command and control of the Amphora is the Octavia controller.

Problem description

Components of the Octavia controller require a shared library that provides the orchestration of create/update/delete actions for Octavia objects such as load balancers and listeners.

It is expected that this library will be used by the Queue Consumer to service API requests, by the Housekeeping Manager to manage the spare Amphora pool, and by the Health Manager to fail over failed objects.

Proposed change

The Controller Worker will be implemented as a class that provides methods to facilitate the create/update/delete actions. This class will be responsible for managing the number of simultaneous operations being executed by coordinating through the Octavia database.

The Controller Worker will provide a base class that sets up and initializes the TaskFlow engines required to complete the action. Users of the library will then call the appropriate method for the action. These methods setup and launch the appropriate flow. Each flow will be contained in a separate class for code reuse and supportability.

The Controller Worker library will provide the following methods:

```
def create_amphora(self):
    """Creates an Amphora.

    :returns: amphora_id
    """
    raise NotImplementedError

def delete_amphora(self, amphora_id):
    """Deletes an existing Amphora.

    :param amphora_id: ID of the amphora to delete
    :returns: None
    :raises AmphoraNotFound: The referenced Amphora was not found
    """
    raise NotImplementedError

def create_load_balancer(self, load_balancer_id):
    """Creates a load balancer by allocating Amphorae.

    :param load_balancer_id: ID of the load balancer to create
    :returns: None
    :raises NoSuitableAmphora: Unable to allocate an Amphora.
    """
    raise NotImplementedError

def update_load_balancer(self, load_balancer_id, load_balancer_updates):
    """Updates a load balancer.

    :param load_balancer_id: ID of the load balancer to update
    :param load_balancer_updates: Dict containing updated load balancer
    attributes
    :returns: None
    :raises LBNotFound: The referenced load balancer was not found
    """
    raise NotImplementedError

def delete_load_balancer(self, load_balancer_id):
    """Deletes a load balancer by de-allocating Amphorae.

    :param load_balancer_id: ID of the load balancer to delete
    :returns: None
    :raises LBNotFound: The referenced load balancer was not found
    """
    raise NotImplementedError

def create_listener(self, listener_id):
    """Creates a listener.

    :param listener_id: ID of the listener to create
    :returns: None
```

(continues on next page)

(continued from previous page)

```
    :raises NoSuitableLB: Unable to find the load balancer
    """
    raise NotImplementedError

def update_listener(self, listener_id, listener_updates):
    """Updates a listener.

    :param listener_id: ID of the listener to update
    :param listener_updates: Dict containing updated listener attributes
    :returns: None
    :raises ListenerNotFound: The referenced listener was not found
    """
    raise NotImplementedError

def delete_listener(self, listener_id):
    """Deletes a listener.

    :param listener_id: ID of the listener to delete
    :returns: None
    :raises ListenerNotFound: The referenced listener was not found
    """
    raise NotImplementedError

def create_pool(self, pool_id):
    """Creates a node pool.

    :param pool_id: ID of the pool to create
    :returns: None
    :raises NoSuitableLB: Unable to find the load balancer
    """
    raise NotImplementedError

def update_pool(self, pool_id, pool_updates):
    """Updates a node pool.

    :param pool_id: ID of the pool to update
    :param pool_updates: Dict containing updated pool attributes
    :returns: None
    :raises PoolNotFound: The referenced pool was not found
    """
    raise NotImplementedError

def delete_pool(self, pool_id):
    """Deletes a node pool.

    :param pool_id: ID of the pool to delete
    :returns: None
    :raises PoolNotFound: The referenced pool was not found
    """
```

(continues on next page)

(continued from previous page)

```
        raise NotImplementedError

    def create_health_monitor(self, health_monitor_id):
        """Creates a health monitor.

        :param health_monitor_id: ID of the health monitor to create
        :returns: None
        :raises NoSuitablePool: Unable to find the node pool
        """
        raise NotImplementedError

    def update_health_monitor(self, health_monitor_id, health_monitor_updates):
        """Updates a health monitor.

        :param health_monitor_id: ID of the health monitor to update
        :param health_monitor_updates: Dict containing updated health monitor
        attributes
        :returns: None
        :raises HMNotFound: The referenced health monitor was not found
        """
        raise NotImplementedError

    def delete_health_monitor(self, health_monitor_id):
        """Deletes a health monitor.

        :param health_monitor_id: ID of the health monitor to delete
        :returns: None
        :raises HMNotFound: The referenced health monitor was not found
        """
        raise NotImplementedError

    def create_member(self, member_id):
        """Creates a pool member.

        :param member_id: ID of the member to create
        :returns: None
        :raises NoSuitablePool: Unable to find the node pool
        """
        raise NotImplementedError

    def update_member(self, member_id, member_updates):
        """Updates a pool member.

        :param member_id: ID of the member to update
        :param member_updates: Dict containing updated member attributes
        :returns: None
        :raises MemberNotFound: The referenced member was not found
        """
        raise NotImplementedError
```

(continues on next page)

(continued from previous page)

```
def delete_member(self, member_id):
    """Deletes a pool member.

    :param member_id: ID of the member to delete
    :returns: None
    :raises MemberNotFound: The referenced member was not found
    """
    raise NotImplementedError

def failover_amphora(self, amphora_id):
    """Failover an amphora

    :param amp_id: ID of the amphora to fail over
    :returns: None
    :raises AmphoraNotFound: The referenced Amphora was not found
    """
    raise NotImplementedError
```

Alternatives

This code could be included in the Queue Consumer component of the controller. However this would not allow the library to be shared with other components of the controller, such as the Health Manager

Data model impact

REST API impact

None

Security impact

Notifications impact

Other end user impact

Performance Impact

Other deployer impact

Developer impact

Implementation

Assignee(s)

Michael Johnson <johnsom>

Work Items

Dependencies

https://blueprints.launchpad.net/octavia/+spec/amphora-driver-interface https://blueprints.launchpad.net/octavia/+spec/neutron-network-driver https://blueprints.launchpad.net/octavia/+spec/nova-compute-driver

Testing

Unit tests

Documentation Impact

None

References

https://blueprints.launchpad.net/octavia/+spec/health-manager https://blueprints.launchpad.net/octavia/+spec/housekeeping-manager https://blueprints.launchpad.net/octavia/+spec/queue-consumer

HAProxy Amphora API

https://blueprints.launchpad.net/octavia/+spec/appliance-api

The reference implementation of Octavia is going to make use of an haproxy- based amphora. As such, there will be an haproxy reference driver that speaks a well-defined protocol to the haproxy-based amphora. This document is meant to be a foundation of this interface, outlining in sufficient detail the various commands that will definitely be necessary. This design should be iterated upon as necessary going forward.

Problem description

This API specification is necessary in order to fully develop the haproxy reference driver, both to ensure this interface is well documented, and so that different people can work on different parts of bringing Octavia to fruition.

Proposed change

Note that this spec does not yet attempt to define the following, though these may follow shortly after this initial spec is approved: * Method for bi-directional authentication between driver and amphora. * Bootstrapping process of amphora * Transition process from "spare" to "active" amphora and other amphora lifecycle transitions

This spec does attempt to provide an initial foundation for the following: * RESTful interface exposed on amphora management

Alternatives

None

Data model impact

None (yet)

REST API impact

Please note that the proposed changes in this spec do NOT affect either the publicly-exposed user or operator APIs, nor really anything above the haproxy reference driver.

Please see doc/main/api/haproxy-amphora-api.rst

Security impact

None yet, though bi-directional authentication between driver and amphora needs to be addressed.

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

stephen-balukoff david-lenwell

Work Items

Dependencies

haproxy reference driver

Testing

Unit tests

Documentation Impact

None

References

None

Housekeeping Manager Specification

https://blueprints.launchpad.net/octavia/+spec/housekeeping-manager

Problem description

The Housekeeping Manager will manage the spare amphora pool and the teardown of amphorae that are no longer needed. On a configurable interval the Housekeeping Manager will check the Octavia database to identify the required cleanup and maintenance actions required. The amphora-lifecycle-management specification details the Create and Deactivate amphora sequences the Housekeeping Manager will follow.

Proposed change

The housekeeping manager will run as a daemon process which will perform the following actions:

- Read the following from the configuration file
  - housekeeping_interval: The time (in seconds) that the housekeeping manager will sleep before running its checks again.
  - spare_amphora_pool_size: The desired number of spare amphorae.
  - maximum_deploying_amphora_count: The maximum number of amphorae that may be deployed simultaneously.
  - maximum_preserved_amphora_count: How many deactivated amphorae to preserve. 0 means delete, 1 or greater means keep up to that many amphorae for future diagnostics. Only amphorae in the ERROR and PRESERVE states are eligible to be preserved. TODO: Right now there is no PRESERVE state, for this to work we would need to define one in the amphora spec.
  - preservation_scheme
    - "keep": keep all preserved amphorae
    - "cycle": maintain a queue of preserved amphorae, deleting the oldest one when a new amphora is preserved.
  - preservation_method: Preservation must take into account the possibility that amphorae instantiated in the future may reuse MAC addresses.
    - "unplug": Disconnect the virtual NICs from the amphora
    - "snapshot": Take a snapshot of the amphora, then stop it
- Get the spare pool size
  - Log the spare pool size
  - If the spare pool size is less than the spare pool target capacity, initiate creation of appropriate number of amphorae.
- Obtain the list of deactivated amphorae and schedule their removal. If preservation_count > 0, and there are fewer than that many amphorae in the preserved pool, preserve the amphora. After the preserved pool size reaches preservation_count, use preservation_scheme to determine whether to keep newly failed amphorae.

- Sleep for the time specified by housekeeping_interval.
- Return to the top

Establish a base class to model the desired functionality:

```
class HousekeepingManager(object):

""" Class to manage the spare amphora pool.  This class should do
very little actual work, its main job is to monitor the spare pool
and schedule creation of new amphrae and removal of used amphrae.
By default, used amphorae will be deleted, but they may optionally
be preserved for future analysis.
"""

    def get_spare_amphora_size(self):
        """ Return the target capacity of the spare pool """
        raise NotImplementedError

    def get_ready_spare_amphora_count(self):
        """ Return the number of available amphorae in the spare pool
        """
        raise NotImplementedError

    def create_amphora(self, num_to_create = 1):
        """ Schedule the creation of the specified number of amphorae
        to be added to the spare pool."""
        raise NotImplementedError

    def remove_amphora(self, amphora_ids):
        """ Schedule the removal of the amphorae specified by
        amphora_ids."""
        raise NotImplementedError
```

Exception Model

The manager is expected to raise or pass along the following well-defined exceptions:

- NotImplementedError - this functionality is not implemented/not supported
- **AmphoraDriverError - a super class for all other exceptions and the catch** all if no specific exception can be found * NotFoundError - this amphora couldn't be found/ was deleted by nova * UnauthorizedException - the driver can't access the amphora * UnavailableException - the amphora is temporary unavailable * DeleteFailed - this load balancer couldn't be deleted

Alternatives

Data model impact

Requires the addition of the housekeeping_interval, spare_pool_size, spare_amphora_pool_size, maximum_preserved_amphora_count, preservation_scheme, and preservation_method to the config.

REST API impact

None.

Security impact

Must follow standard practices for database access.

Notifications impact

Other deployer impact

Other end user impact

There should be no end-user-visible impact.

Performance Impact

The housekeeping_interval and spare_pool_size parameters will be adjustable by the operator in order to balance resource usage against performance.

Developer impact

Developers of other modules need to be aware that amphorae may be created, deleted, or saved for diagnosis by this daemon.

Implementation

Assignee(s)

Al Miller <ajmiller>

Work Items

- Write abstract interface
- Write Noop driver
- Write tests

Dependencies

Amphora driver Config manager

Testing

- Unit tests with tox and Noop-Driver
- tempest tests with Noop-Driver

Documentation Impact

None - we won't document the interface for 0.5. If that changes we need to write an interface documentation so 3rd party drivers know what we expect.

References

Network Driver Interface

Include the URL of your launchpad blueprint:

https://blueprints.launchpad.net/octavia/+spec/network-driver-interface

We need a generic interface in which to create networking resources. This is to allow implementations that can support different networking infrastructures that accomplish frontend and backend connectivity.

Problem description

There is a need to define a generic interface for a networking service. An Octavia controller should not know what networking infrastructure is being used underneath. It should only know an interface. This interface is needed to support differing networking infrastructures.

Proposed change

In order to make the network driver as genericly functional as possible, it is broken down into methods that Octavia will need at a high level to accomplish frontend and backend connectivity. The idea is that to implement these methods it may require multiple requests to the networking service to accomplish the end result. The interface is meant to promote stateless implementations and suffer no issues being run in parallel.

In the future we would like to create a common module that implementations of this interface can call to setup a taskflow engine to promote using a common taskflow configuration. That however, can be left once this has had time to mature.

Existing data model:

- **class VIP**
  - load_balancer_id
  - ip_address
  - network_id - (neutron subnet)
  - port_id - (neutron port)
- **class Amphora**
  - load_balancer_id
  - compute_id
  - lb_network_ip
  - status
  - **vrrp_ip - if an active/passive topology, this is the ip where the vrrp** communication between peers happens
  - **ha_ip - this is the highly available IP. In an active/passive topology** it most likely exists on the MASTER amphora and on failure it will be raised on the BACKUP amphora. In an active/active topology it may exist on both amphorae. In the end, it is up to the amphora driver to decide how to use this.

New data models:

- **class Interface**
  - id
  - network_id - (neutron subnet)
  - amphora_id
  - fixed_ips
- **class Delta**
  - amphora_id
  - compute_id
  - add_nics
  - delete_nics

- **class Network**
    - id
    - name
    - subnets - (list of subnet ids)
    - tenant_id
    - admin_state_up
    - provider_network_type
    - provider_physical_network
    - provider_segmentation_id
    - router_external
    - mtu
- **class Subnet**
    - id
    - name
    - network_id
    - tenant_id
    - gateway_ip
    - cidr
    - ip_version
- **class Port**
    - id
    - name
    - device_id
    - device_owner
    - mac_address
    - network_id
    - status
    - tenant_id
    - admin_state_up
    - fixed_ips - list of FixedIP objects
- **FixedIP**
    - subnet_id
    - ip_address
- **AmphoraNetworkConfig**

    - amphora - Amphora object
    - vip_subnet - Subnet object
    - vip_port - Port object
    - vrrp_subnet - Subnet object
    - vrrp_port - Port object
    - ha_subnet - Subnet object
    - ha_port - Port object

New Exceptions defined in the octavia.network package:

- NetworkException - Base Exception
- PlugVIPException
- UnplugVIPException
- PluggedVIPNotFound
- AllocateVIPException
- DeallocateVIPException
- PlugNetworkException
- UnplugNetworkException
- VIPInUse
- PortNotFound
- SubnetNotFound
- NetworkNotFound
- AmphoraNotFound

This class defines the methods for a fully functional network driver. Implementations of this interface can expect a rollback to occur if any of the non-nullipotent methods raise an exception.

class AbstractNetworkDriver

- plug_vip(loadbalancer, vip)
  - Sets up the routing of traffic from the vip to the load balancer and its amphorae.
  - loadbalancer - instance of data_models.LoadBalancer
    - this is to keep the parameters as generic as possible so different implementations can use different properties of a load balancer. In the future we may want to just take in a list of amphora compute ids and the vip data model.
  - vip = instance of a VIP
  - returns list of Amphora
  - raises PlugVIPException, PortNotFound
- unplug_vip(loadbalancer, vip)
  - Removes the routing of traffic from the vip to the load balancer and its amphorae.

  - loadbalancer = instance of a data_models.LoadBalancer
  - vip = instance of a VIP
  - returns None
  - raises UnplugVIPException, PluggedVIPNotFound
- allocate_vip(loadbalancer)
  - Allocates a virtual ip and reserves it for later use as the frontend connection of a load balancer.
  - loadbalancer = instance of a data_models.LoadBalancer
  - returns VIP instance
  - raises AllocateVIPException, PortNotFound, SubnetNotFound
- deallocate_vip(vip)
  - Removes any resources that reserved this virtual ip.
  - vip = VIP instance
  - returns None
  - raises DeallocateVIPException, VIPInUse
- plug_network(compute_id, network_id, ip_address=None)
  - Connects an existing amphora to an existing network.
  - compute_id = id of an amphora in the compute service
  - network_id = id of the network to attach
  - ip_address = ip address to attempt to be assigned to interface
  - returns Interface instance
  - raises PlugNetworkException, AmphoraNotFound, NetworkNotFound
- unplug_network(compute_id, network_id, ip_address=None)
  - Disconnects an existing amphora from an existing network. If ip_address is not specified then all interfaces on that network will be unplugged.
  - compute_id = id of an amphora in the compute service to unplug
  - network_id = id of network to unplug amphora
  - ip_address = ip address of interface to unplug
  - returns None
  - **raises UnplugNetworkException, AmphoraNotFound, NetworkNotFound,** NetworkException
- get_plugged_networks(compute_id):
  - Retrieves the current plugged networking configuration
  - compute_id = id of an amphora in the compute service
  - returns = list of Instance instances
- update_vip(loadbalancer):

  - Hook for the driver to update the VIP information based on the state of the passed in loadbalancer
  - loadbalancer: instance of a data_models.LoadBalancer
- get_network(network_id):
  - Retrieves the network from network_id
  - network_id = id of an network to retrieve
  - returns = Network data model
  - raises NetworkException, NetworkNotFound
- get_subnet(subnet_id):
  - Retrieves the subnet from subnet_id
  - subnet_id = id of a subnet to retrieve
  - returns = Subnet data model
  - raises NetworkException, SubnetNotFound
- get_port(port_id):
  - Retrieves the port from port_id
  - port_id = id of a port to retrieve
  - returns = Port data model
  - raises NetworkException, PortNotFound
- failover_preparation(amphora):
  - Prepare an amphora for failover
  - amphora = amphora data model
  - returns = None
  - raises PortNotFound

Alternatives

- Straight Neutron Interface (networks, subnets, ports, floatingips)
- Straight Nova-Network Interface (network, fixed_ips, floatingips)

Data model impact

- The Interface data model defined above will just be a class. We may later decide that it needs to be stored in the database, but we can optimize on that in a later review if needed.

REST API impact

None

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

Need a service account to own the resources these methods create.

Developer impact

This will be creating an interface in which other code will be creating network resources.

Implementation

Assignee(s)

brandon-logan

Work Items

Define interface

Dependencies

None

Testing

None

Documentation Impact

Just docstrings on methods.

References

None

Nova Compute Driver

Blueprint: https://blueprints.launchpad.net/octavia/+spec/nova-compute-driver

Octavia needs to interact with nova for creation of VMs for this version. This spec will flesh out all the methods described in the compute-driver-interface with nova VM specific commands.

Problem description

This spec details operations for creating, updating, and modifying amphora that will hold the actual load balancer. It will utilize the nova client python api version 3 for the nova specific requests and commands.

Proposed change

Expose nova operations

- Build: Will need to build a virtual machine according to configuration parameters
  - Will leverage the nova client ServerManager method "create" to build a server
- Get: Will need to retrieve details of the virtual machine from nova
  - Will leverage the nova client ServerManager method "get" to retrieve a server, and return an amphora object
- Delete: Will need to remove a virtual machine
  - Will leverage the nova client ServerManager method "delete" for removal of server
- Status: Will need to retrieve the status of the virtual machine
  - Will leverage the aforementioned get call to retrieve status of the server

Alternatives

None

Data model impact

Add fields to existing Amphora object

REST API impact

None

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

Will need a nova service account and necessary credentials stored in config

Implementation

Assignee(s)

trevor-vardeman

Work Items

Expose nova operations

Dependencies

compute-driver-interface

Testing

Unit tests Functional tests

Documentation Impact

None

References

https://blueprints.launchpad.net/octavia/+spec/nova-compute-driver https://docs.openstack.org/python-novaclient/latest/reference/api/index.html

Octavia Operator API Foundation

https://blueprints.launchpad.net/octavia/+spec/operator-api

Octavia needs the foundation of the Operator API created. This spec is not meant to address every functionality needed in the operator API, only to create a solid foundation to iterate on in the future.

Problem description

This is needed because this will be the mechanism to actually communicate with Octavia. Doing CRUD operations on all entities will be needed ASAP so that the system can be thoroughly tested.

Proposed change

Expose Pecan resources - Defined explicitly below in the REST API Impact

Create WSME types - These will be responsible for request validation and deserialization, and also response serialization

Setup paste deploy - This will be used in the future to interact with keystone and other middleware, however at first this will not have any authentication so tenant_ids will just have to be made up uuids.

Create a handler interface and a noop logging implementation - A handler interface will be created. This abstraction layer is needed because calling the controller in the resource layer will work for 0.5 but 1.0 will be sending it off to a queue. With this abstraction layer we can easily swap out a 0.5 controller with a 1.0 controller.

Call database repositories - Most if not all resources will make a call to the database

Call handler - Only create, update, and delete operations should call the handler

Alternatives

None

Data model impact

Will need to add some methods to the database repository

REST API impact

Exposed Resources and Methods

POST /loadbalancers * Successful Status Code - 202 * JSON Request Body Attributes ** vip - another JSON object with one required attribute from the following *** net_port_id - uuid *** subnet_id - uuid *** floating_ip_id - uuid *** floating_ip_network_id - uuid ** tenant_id - string - optional - default "0" * 36 (for now) ** name - string - optional - default null ** description - string - optional - default null ** enabled - boolean - optional - default true * JSON Response Body Attributes ** id - uuid ** vip - another JSON object *** net_port_id - uuid *** subnet_id - uuid *** floating_ip_id - uuid *** floating_ip_network_id - uuid ** tenant_id - string ** name - string ** description - string ** enabled - boolean ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

PUT /loadbalancers/{lb_id} * Successful Status Code - 202 * JSON Request Body Attributes ** name - string ** description - string ** enabled - boolean * JSON Response Body Attributes ** id - uuid ** vip - another JSON object *** net_port_id - uuid *** subnet_id - uuid *** floating_ip_id - uuid *** floating_ip_network_id - uuid ** tenant_id - string ** name - string ** description - string ** enabled - boolean ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

DELETE /loadbalancers/{lb_id} * Successful Status Code - 202 * No response or request body

GET /loadbalancers/{lb_id} * Successful Status Code - 200 * JSON Response Body Attributes ** id - uuid ** vip - another JSON object **net_port_id - uuid** subnet_id - uuid **floating_ip_id - uuid** floating_ip_network_id - uuid ** tenant_id - string ** name - string ** description - string ** enabled - boolean ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

GET /loadbalancers?tenant_id * Successful Status Code - 200 * tenant_id is an optional query parameter to filter by tenant_id * returns a list of load balancers

POST /loadbalancers/{lb_id}/listeners * Successful Status Code - 202 * JSON Request Body Attributes ** protocol - string enum - (TCP, HTTP, HTTPS) - required ** protocol_port - integer - required ** connection_limit - integer - optional ** default_tls_container_id - uuid - optional ** tenant_id - string - optional - default "0" * 36 (for now) ** name - string - optional - default null ** description - string - optional - default null ** enabled - boolean - optional - default true * JSON Response Body Attributes ** id - uuid ** protocol - string enum - (TCP, HTTP, HTTPS) ** protocol_port - integer ** connection_limit - integer ** default_tls_container_id - uuid ** tenant_id - string - optional ** name - string - optional ** description - string - optional ** enabled - boolean - optional ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

PUT /loadbalancers/{lb_id}/listeners/{listener_id} * Successful Status Code - 202 * JSON Request Body Attributes ** protocol - string enum ** protocol_port - integer ** connection_limit - integer ** default_tls_container_id - uuid ** name - string ** description - string ** enabled - boolean * JSON Response Body Attributes ** id - uuid ** protocol - string enum - (TCP, HTTP, HTTPS) ** protocol_port - integer ** connection_limit - integer ** default_tls_container_id - uuid ** tenant_id - string - optional ** name - string - optional ** description - string - optional ** enabled - boolean - optional ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

DELETE /loadbalancers/{lb_id}/listeners/{listener_id} * Successful Status Code - 202 * No response or request body

GET /loadbalancers/{lb_id}/listeners/{listener_id} * Successful Status Code - 200 * JSON Response Body Attributes ** id - uuid ** protocol - string enum - (TCP, HTTP, HTTPS) ** protocol_port - integer ** connection_limit - integer ** default_tls_container_id - uuid ** tenant_id - string - optional ** name - string - optional ** description - string - optional ** enabled - boolean - optional ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

GET /loadbalancers/{lb_id}/listeners * Successful Status Code - 200 * A list of listeners on load balancer lb_id

POST /loadbalancers/{lb_id}/listeners/{listener_id}/pools * Successful Status Code - 202 * JSON Request Body Attributes ** protocol - string enum - (TCP, HTTP, HTTPS) - required ** lb_algorithm - string enum - (ROUND_ROBIN, LEAST_CONNECTIONS, RANDOM) - required ** session_persistence - JSON object - optional **type - string enum - (SOURCE_IP, HTTP_COOKIE) - required** cookie_name - string - required for HTTP_COOKIE type ** tenant_id - string - optional - default "0" * 36 (for now) ** name - string - optional - default null ** description - string - optional - default null ** enabled - boolean - optional - default true * JSON Response Body Attributes ** id - uuid ** protocol - string enum - (TCP, HTTP, HTTPS) ** lb_algorithm - string enum - (ROUND_ROBIN, LEAST_CONNECTIONS, RANDOM) ** session_persistence - JSON object **type - string enum - (SOURCE_IP, HTTP_COOKIE)** cookie_name - string ** name - string ** description - string ** enabled - boolean ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

PUT /loadbalancers/{lb_id}/listeners/{listener_id}/pools/{pool_id} * Successful Status Code - 202 * JSON Request Body Attributes ** protocol - string enum - (TCP, HTTP, HTTPS) ** lb_algorithm - string enum - (ROUND_ROBIN, LEAST_CONNECTIONS, RANDOM) ** session_persistence - JSON object *** type - string enum - (SOURCE_IP, HTTP_COOKIE) *** cookie_name - string ** name - string ** description - string ** enabled - boolean * JSON Response Body Attributes ** id - uuid ** protocol - string enum - (TCP, HTTP, HTTPS) ** lb_algorithm - string enum - (ROUND_ROBIN, LEAST_CONNECTIONS, RANDOM) ** session_persistence - JSON object *** type - string enum - (SOURCE_IP, HTTP_COOKIE) *** cookie_name - string ** name - string ** description - string ** enabled - boolean ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

DELETE /loadbalancers/{lb_id}/listeners/{listener_id}/pools/{pool_id} * Successful Status Code - 202 No request or response body

GET /loadbalancers/{lb_id}/listeners/{listener_id}/pools/{pool_id} * Successful Status Code - 200 * JSON Response Body Attributes ** id - uuid ** protocol - string enum - (TCP, HTTP, HTTPS) ** lb_algorithm - string enum - (ROUND_ROBIN, LEAST_CONNECTIONS, RANDOM) ** session_persistence - JSON object *** type - string enum - (SOURCE_IP, HTTP_COOKIE) *** cookie_name - string ** name - string ** description - string ** enabled - boolean ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

GET /loadbalancers/{lb_id}/listeners/{listener_id}/pools * Successful Status Code - 200 * Returns a list of pools

POST /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/healthmonitor * Successful Status Code - 202 * JSON Request Body Attributes ** type - string enum - (HTTP, HTTPS, TCP) - required ** delay - integer - required ** timeout - integer - required ** fall_threshold - integer - required ** rise_threshold - integer - required ** http_method - string enum - (GET, POST, PUT, DELETE) - required for HTTP(S) ** url_path - string - required for HTTP(S) ** expected_codes - comma delimited string - required for HTTP(S) ** enabled - boolean - required - default true * JSON Response Body Attributes ** type - string enum - (HTTP, HTTPS, TCP) ** delay - integer ** timeout - integer ** fall_threshold - integer ** rise_threshold - integer ** http_method - string enum - (GET, POST, PUT, DELETE) ** url_path - string ** expected_codes - comma delimited string ** enabled - boolean

PUT /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/healthmonitor * Successful Status Code - 202 * JSON Request Body Attributes ** type - string enum - (HTTP, HTTPS, TCP) ** delay - integer ** timeout - integer ** fall_threshold - integer ** rise_threshold - integer ** http_method - string enum - (GET, POST, PUT, DELETE) ** url_path - string ** expected_codes - comma delimited string ** enabled - boolean * JSON Response Body Attributes ** type - string enum - (HTTP, HTTPS, TCP) ** delay - integer ** timeout - integer ** fall_threshold - integer ** rise_threshold - integer ** http_method - string enum - (GET, POST, PUT, DELETE) ** url_path - string ** expected_codes - comma delimited string ** enabled - boolean

DELETE /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/healthmonitor * Successful Status Code - 202 No request or response body

GET /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/healthmonitor * Successful Status Code - 200 * JSON Response Body Attributes ** type - string enum - (HTTP, HTTPS, TCP) ** delay - integer ** timeout - integer ** fall_threshold - integer ** rise_threshold - integer ** http_method - string enum - (GET, POST, PUT, DELETE) ** url_path - string ** expected_codes - comma delimited string ** enabled - boolean

POST /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/members * Successful Status Code - 202 * JSON Request Body Attributes ** ip_address - IP Address - required ** protocol_port - integer - required ** weight - integer - optional ** subnet_id - uuid - optional ** tenant_id - string -

optional - default "0" * 36 (for now) ** enabled - boolean - optional - default true * JSON Response Body Attributes ** id - uuid ** ip_address - IP Address ** protocol_port - integer ** weight - integer ** subnet_id - uuid ** tenant_id - string ** enabled - boolean ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

PUT /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/members/{member_id} * Successful Status Code - 202 * JSON Request Body Attributes ** protocol_port - integer - required ** weight - integer - optional ** enabled - boolean - optional - default true * JSON Response Body Attributes ** id - uuid ** ip_address - IP Address ** protocol_port - integer ** weight - integer ** subnet_id - uuid ** tenant_id - string ** enabled - boolean ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

DELETE /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/members/{member_id} * Successful Status Code - 202 No request or response body

GET /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/members/{member_id} * Successful Status Code - 200 * JSON Response Body Attributes ** id - uuid ** ip_address - IP Address ** protocol_port - integer ** weight - integer ** subnet_id - uuid ** tenant_id - string ** enabled - boolean ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR)

GET /loadbalancers/{lb_id}/listeners/{listener_id}/ pools/{pool_id}/members * Successful Status Code - 200 Returns a list of members

Security impact

No authentication with keystone

Notifications impact

None

Other end user impact

Not ready for end user

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

brandon-logan

Work Items

Expose Pecan resources Create WSME types Setup paste deploy Create a handler interface and a noop logging implementation Call database repositories Call handler

Dependencies

db-repositories

Testing

Unit tests

Documentation Impact

None

References

None

Queue Consumer

https://blueprints.launchpad.net/octavia/+spec/queue-consumer

This blueprint describes how Oslo messages are consumed, processed and delegated from the API-controller queue to the controller worker component of Octavia. The component that is responsible for these activities is called the Queue Consumer.

Problem description

Oslo messages need to be consumed by the controller and delegated to the proper controller worker. Something needs to interface with the API-controller queue and spawn the controller workers. That "something" is what we are calling the Queue Consumer.

Proposed change

The major component of the Queue Consumer will be a class that acts as a consumer to Oslo messages. It will be responsible for configuring and starting a server that is then able to receive messages. There will be a one-to-one mapping between API methods and consumer methods (see code snippet below). Corresponding controller workers will be spawned depending on which consumer methods are called.

The threading will be handled by Oslo messaging using the 'eventlet' executor. Using the 'eventlet' executor will allow for message throttling and removes the need for the controller workers to manage threads. The benefit of using the 'eventlet' executor is that the Queue Consumer will not have to spawn threads at all, since every message received will be in its own thread already. This means that the Queue Consumer doesn't spawn a controller worker, rather it just starts the execution of the deploy code.

An 'oslo_messaging' configuration section will need to be added to octavia.conf for Oslo messaging options. For the Queue Consumer, the 'rpc_thread_pool_size' config option will need to be added. This option will determine how many consumer threads will be able to read from the queue at any given time (per consumer instance) and serve as a throttling mechanism for message consumption. For example, if 'rpc_thread_pool_size' is set to 1 thread then only one controller worker will be able to conduct work. When that controller worker completes its task then a new message can be consumed and a new controller worker flow started.

Below are the planned interface methods for the queue consumer. The Queue Consumer will be listening on the **OCTAVIA_PROV** (short for octavia provisioning) topic. The *context* parameter will be supplied along with an identifier such as a load balancer id, listener id, etc. relevant to the particular interface method. The *context* parameter is a dictionary and is reserved for metadata. For example, the Neutron LBaaS agent leverages this parameter to send additional request information. Additionally, update methods include a **_updates* parameter than includes the changes that need to be made. Thus, the controller workers responsible for the update actions will need to query the database to retrieve the old state and combine it with the updates to provision appropriately. If a rollback or exception occur, then the controller worker will only need to update the provisioning status to **ERROR** and will not need to worry about making database changes to attributes of the object being updated.

```
def create_load_balancer(self, context, load_balancer_id):
    pass

def update_load_balancer(self, context, load_balancer_updates,
                         load_balancer_id):
    pass

def delete_load_balancer(self, context, load_balancer_id):
    pass

def create_listener(self, context, listener_id):
    pass

```

(continues on next page)

(continued from previous page)

```
def update_listener(self, context, listener_updates, listener_id):
    pass

def delete_listener(self, context, listener_id):
    pass

def create_pool(self, context, pool_id):
    pass

def update_pool(self, context, pool_updates, pool_id):
    pass

def delete_pool(self, context, pool_id):
    pass

def create_health_monitor(self, context, health_monitor_id):
    pass

def update_health_monitor(self, context, health_monitor_updates,
                          health_monitor_id):
    pass

def delete_health_monitor(self, context, health_monitor_id):
    pass

def create_member(self, context, member_id):
    pass

def update_member(self, context, member_updates, member_id):
    pass

def delete_member(self, context, member_id):
    pass
```

Alternatives

There are a variety of ways to consume from Oslo messaging. For example, instead of having a single consumer on the controller we could have multiple consumers (i.e. one for CREATE messages, one for UPDATE messages, etc.). However, since we merely need something to pass messages off to controller workers other options are overkill.

Data model impact

While there is no direct data model impact it is worth noting that the API will not be persisting updates to the database. Rather, delta updates will pass from the user all the way to the controller worker. Thus, when the controller worker successfully completes the prescribed action, only then will it persist the updates to the database. No API changes are necessary for create and update actions.

REST API impact

None

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

The only performance related item is queue throttling. This is done by design so that operators can safely throttle incoming messages dependent on their specific needs.

Other deployer impact

Configuration options will need to be added to ocativa.conf. Please see above for more details.

Developer impact

None

Implementation

Assignee(s)

jorge-miramontes

Work Items

- Implement consumer class
- Add executable queue-consumer.py to bin directory

Dependencies

https://blueprints.launchpad.net/octavia/+spec/controller-worker

Testing

Unit tests

Documentation Impact

None

References

None

TLS Data Security and Barbican

Launchpad blueprint:

https://blueprints.launchpad.net/octavia/+spec/tls-data-security

Octavia will have some need of secure storage for TLS related data. This BP is intended to identify all of the data that needs secure storage, or any other interaction that will require the use of Barbican or another secure solution.

Problem description

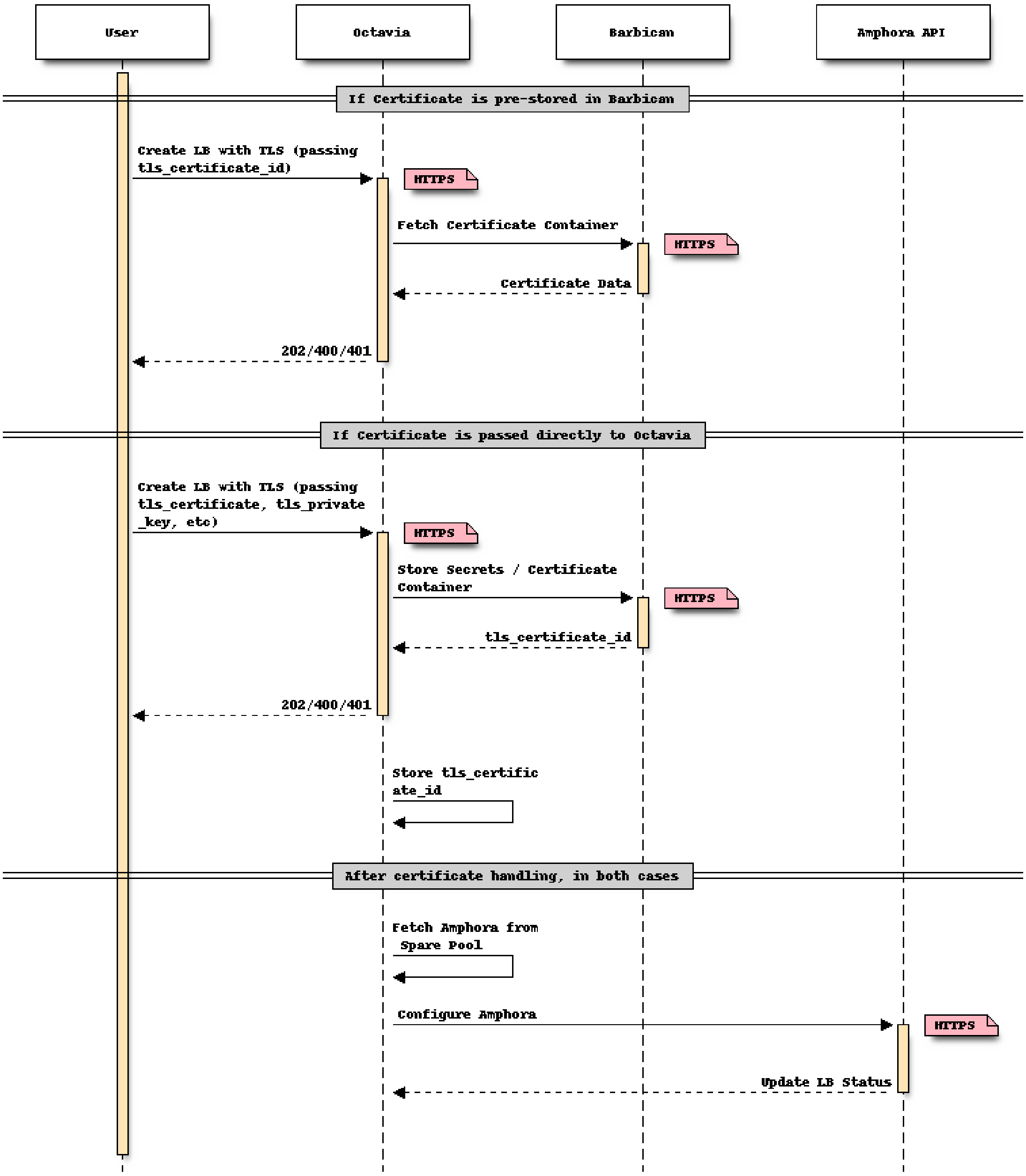
1. Octavia will support TLS Termination (including SNI), which will require us to store and retrieve certificates and private keys from a secure repository.

2. Octavia will communicate with its Amphorae using TLS, so each Amphora will need a certificate for the controller to validate.

3. Octavia will need TLS data for exposing its own API via HTTPS.

Proposed change

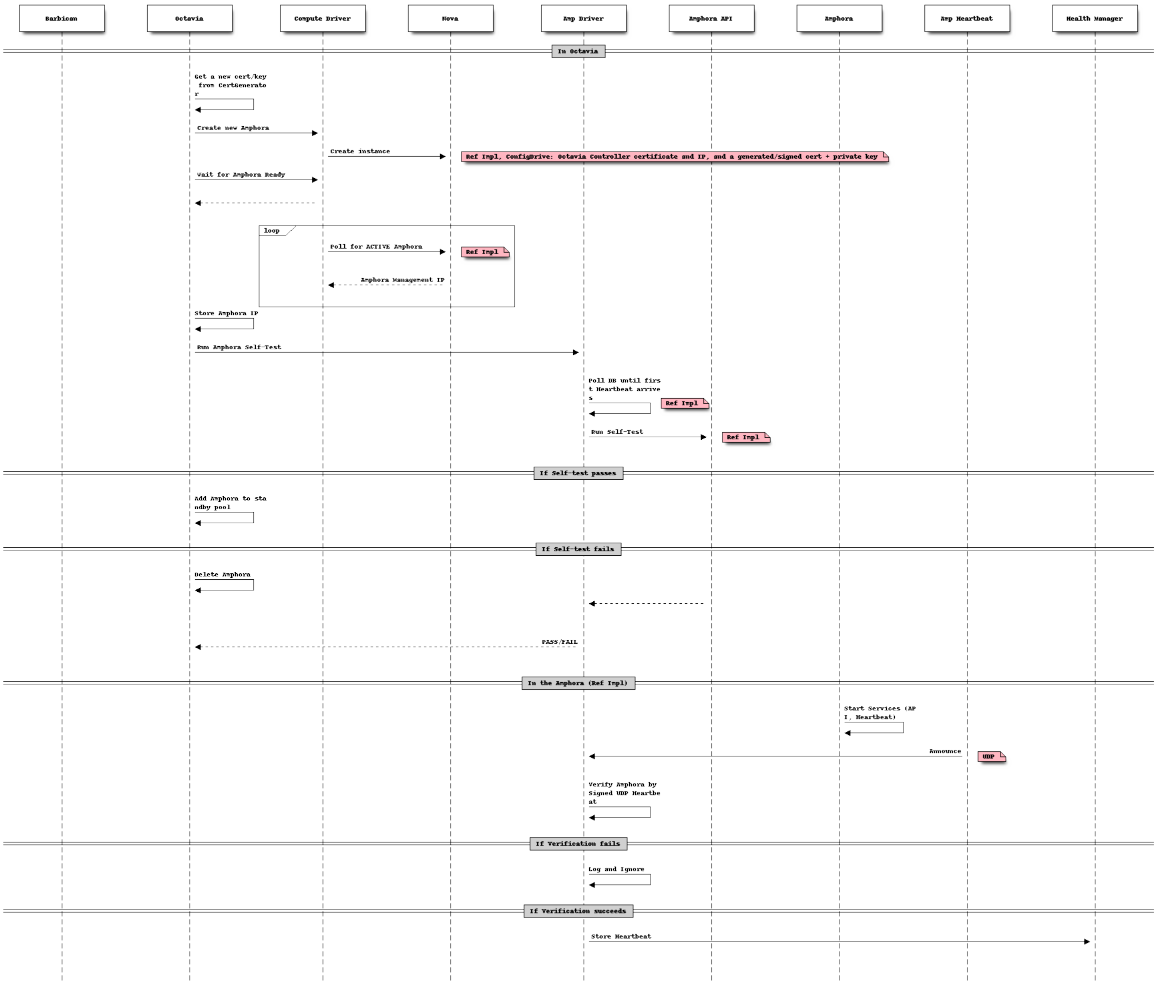
The initial supported implementation for TLS related functions will be Barbican, but the interface will be generic such that other implementations could be created later.


User
Octavia
Barbican
Amphora API
If Certificate is pre-stored in Barbican
Create LB with TLS (passing tls_certificate_id)
HTTPS
Fetch Certificate Container
HTTPS
Certificate Data
202/400/401
If Certificate is passed directly to Octavia
Create LB with TLS (passing tls_certificate, tls_private _key, etc)
HTTPS
Store Secrets / Certificate Container
HTTPS
tls_certificate_id
202/400/401
Store tls_certific ate_id
After certificate handling, in both cases
Fetch Amphora from Spare Pool
Configure Amphora
HTTPS
Update LB Status

1. Create a CertificateManager interface for storing and retrieving certificate and private key pairs (and

intermediate certs / private key passphrase). Users will pass their TLS data to Octavia in the form of a certificate_id, which is a reference to their data in some secure service. Octavia will store that certificate_id for each Listener/SNI and will retrieve the data when necessary. (Barbican specific: users will need to add Octavia's user account as an authorized user on the Container and all Secrets [1] so we catch fetch the data on their behalf.)

We will need to validate the certificate data (including key and intermediates) when we initially receive it, and will assume that it remains unchanged for the lifespan of the LB (in Barbican the data is immutable so this is a safe assumption, I do not know how well this will work for other services). In the case of invalid TLS data, we will reject the request with a 400 (if it is an initial create) or else put the LB into ERROR status (if it is on a failover event or during some other non-interactive scenario).


Barbican
Octavia
Compute Driver
Nova
Amp Driver
Amphora API
Amphora
Amp Heartbeat
Health Manager
In Octavia
Get a new cert/key from CertGenerator
Create new Amphora
Create instance
Ref Impl, ConfigDrive: Octavia Controller certificate and IP, and a generated/signed cert + private key
Wait for Amphora Ready
loop
Poll for ACTIVE Amphora
Ref Impl
Amphora Management IP
Store Amphora IP
Run Amphora Self-Test
Poll DB until first Heartbeat arrives
Ref Impl
Run Self-Test
Ref Impl
If Self-test passes
Add Amphora to standby pool
If Self-test fails
Delete Amphora
PASS/FAIL
In the Amphora (Ref Impl)
Start Services (API, Heartbeat)
Announce
UDP
Verify Amphora by Signed UDP Heartbeat
If Verification fails
Log and Ignore
If Verification succeeds
Store Heartbeat

2. Create a CertificateGenerator interface to generate certificates from CSRs. When the controller creates an Amphora, it will generate a private key and a CSR, generate a signed certificate from the CSR, and include the private key and signed certificate in a ConfigDrive for the new Amphora. It will also include a copy of the Controller's certificate on the ConfigDrive. All future communications with the Amphora will do certificate validation based on these certificates. For the Amphora, this will be based on our (private) certificate authority and the CN of the Amphora's cert matching the ID of the Amphora. For the Controller, the cert should be a complete match with the version provided.

(The CertificateManager and CertificateGenerator interfaces are separate because while Barbican can perform both functions, future implementations may need to use two distinct services to achieve both.)

3. The key/cert for the main Octavia API/controller should be maintained manually by the server operators using whatever configuration management they choose. We should not need to use a specific external

repo for this. The trusted CA Cert will also need to be retrieved from barbican and manually loaded in the config.

Alternatives

We could skip the interface and just use Barbican directly, but that would be diverging from what seems to be the accepted OpenStack model for Secret Store integration.

We could also store everything locally or in the DB, but that isn't a real option for production systems because it is incredibly insecure (though there will be a "dummy driver" that operates this way for development purposes).

Data model impact

Nothing new, the models for this should already be in place. Some of the columns/classes might need to be renamed more generically (currently there is a tls_container_id column, which would become tls_certificate_id to be more generic).

REST API impact

None

Security impact

Using Barbican is considered secure.

Notifications impact

None

Other end user impact

None

Performance Impact

Adding an external touchpoint (a certificate signing service) to the Amphora spin-up workflow will increase the average time for readying an Amphora. This shouldn't be a huge problem if the standby-pool size is sufficient for the particular deployment.

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

Adam Harwell (adam-harwell)

Work Items

1. Create CertificateManager interface.
2. Create CertificateGenerator interface.
3. Create BarbicanCertificateManager implementation.
4. Create BarbicanCertificateGenerator implementation.
5. Create unit tests!

Dependencies

This script will depend on the OpenStack Barbican project, including some features that are still only at the blueprint stage.

Testing

There will be testing. Yes.

Documentation Impact

Documentation changes will be primarily internal.

References

[1] https://review.opendev.org/#/c/127353/

[2] https://review.opendev.org/#/c/129048/

4.5.2 Version 0.8 (mitaka)

Active-Standby Amphora Setup using VRRP

https://blueprints.launchpad.net/octavia/+spec/activepassiveamphora

This blueprint describes how Octavia implements its Active/Standby solution. It will describe the high level topology and the proposed code changes from the current supported Single topology to realize the high availability loadbalancer scenario.

Problem description

A tenant should be able to start high availability loadbalancer(s) for the tenant's backend services as follows:

- The operator should be able to configure an Active/Standby topology through an octavia configuration file, which the loadbalancer shall support. An Active/Standby topology shall be supported by Octavia in addition to the Single topology that is currently supported.
- In Active/Standby, two Amphorae shall host a replicated configuration of the load balancing services. Both amphorae will also deploy a Virtual Router Redundancy Protocol (VRRP) implementation [2].
- Upon failure of the master amphora, the backup one shall seamlessly take over the load balancing functions. After the master amphora changes to a healthy status, the backup amphora shall give up the load balancing functions to the master again (see [2] section 3 for details on master election protocol).
- Fail-overs shall be seamless to end-users and fail-over time should be minimized.
- The following diagram illustrates the Active/Standby topology.

asciiflow:

```
+--------+
| Tenant |
|Service |
|  (1)   |
+--------+        +-----------+
| +--------+ +----+  Master   +----+
| | Tenant | |VIP |  Amphora  |IP1 |
| |Service | +--+-+-----+-----+-+--+
| |   (M)  |    | |MGMT |VRRP | |
| +--------+    | | IP  | IP1 | |
| |  Tenant     | +--+--++----+ |
| | Network     |    |   |      |   +-----------------+ Floating +---------+
v-v-------------^----+---v-^----v-^-+     Router      |    IP    |         |
```

(continues on next page)

(continued from previous page)

```
^---------------+----v-^---+------+-+Floating <-> VIP <----------+ Internet|
|  Management   |      |   |      | |                 |          |         |
|    (MGMT)     |      |   |      | +-----------------+          +---------+
|   Network     |   +--+--++----+ |
|           Paired  |MGMT |VRRP | |
|               |   | IP  | IP2 | |
+-----------+   |   +-----+-----+ |
|  Octavia   |  ++---+  Backup   +-+--+
|Controller |  |VIP |  Amphora  |IP2 |
|    (s)    |  +----+-----------+----+
+-----------+
```

- The newly introduced VRRP IPs shall communicate on the same tenant network (see security impact for more details).
- The existing Haproxy Jinja configuration template shall include "peer" setup for state synchronization over the VRRP IP addresses.
- The VRRP IP addresses shall work with both IPv4 and IPv6.

Proposed change

The Active/Standby loadbalancers require the following high level changes:

- Add support of VRRP in the amphora base image through Keepalived.
- Extend the controller worker to be able to spawn N amphorae associated with the same loadbalancer on N different compute nodes (This takes into account future work on Active/Active topology). The amphorae shall be allowed to use the VIP through "allow address pairing". These amphorae shall replicate the same listeners, and pools configuration. Note: topology is a property of a load balancer and not of one of its amphorae.
- Extend the amphora driver interface, the amphora REST driver, and Jinja configuration templates for the newly introduced VRRP service [4].
- Develop a Keepalived driver.
- Extend the network driver to become aware of the different loadbalancer topologies and add support of network creation. The network driver shall also pair the different amphorae in a given topology to the same VIP address.
- Extend the controller worker to build the right flow/sub-flows according to the given topology. The controller worker is also responsible of creating the correct stores needed by other flow/sub-flows.
- Extend the Octavia configuration and Operator API to support the Active/Standby topology.
- MINOR: Extend the Health Manager to be aware of the role of the amphora (Master/Backup) [9]. If the health manager decided to spawn a new amphora to replace an unhealthy one (while a backup amphora is already in service), it must replicate the same VRRP priorities, ids, and authentication credentials to keep the loadbalancer in its appropriate configuration. Listeners associated with this load balancer shall be put in a DEGRADED provisioning state.

Alternatives

We could use heartbeats as an alternative to VRRP, which is also a widely adopted solution. Heartbeats better suit redundant file servers, filesystems, and databases rather than network services such as routers, firewalls, and loadbalancers. Willy Tarreau, the creator of Haproxy, provides a detailed view on the major differences between heartbeats and VRRP in [5].

Data model impact

The data model of the Octavia database shall be impacted as follows:

- A new column in the load_balancer table shall indicate its topology. The topology field takes values from: SINGLE, or ACTIVE/STANDBY.
- A new column in the amphora table shall indicate an amphora's role in the topology. If the topology is SINGLE, the amphora role shall be STANDALONE. If the topology is ACTIVE/STANDBY, the amphora role shall be either MASTER or BACKUP. This role field will also be of use for the Active/Active topology.
- New value tables for the loadbalancer topology and the amphorae roles.
- New columns in the amphora table shall indicate the VRRP priority, the VRRP ID, and the VRRP interface of the amphora.
- A new column in the listener table shall indicate the TCP port used for listener internal data synchronization.
- VRRP groups define the common VRRP configurations for all listeners on an amphora. A new table shall hold the VRRP groups main configuration primitives including at least: VRRP authentication information, role and priority advertisement interval. Each Active/Standby loadbalancer defines one and only one VRRP group.

REST API impact

** Changes to amphora API: see [11] **

PUT /listeners/{amphora_id}/{listener_id}/haproxy

PUT /vrrp/upload

PUT /vrrp/{action}

GET /interface/{ip_addr}

** Changes to operator API: see [10] **

POST /loadbalancers * Successful Status Code - 202 * JSON Request Body Attributes ** vip - another JSON object with one required attribute from the following *** net_port_id - uuid *** subnet_id - uuid *** floating_ip_id - uuid *** floating_ip_network_id - uuid ** tenant_id - string - optional - default "0" * 36 (for now) ** name - string - optional - default null ** description - string - optional - default null ** enabled - boolean - optional - default true * JSON Response Body Attributes ** id - uuid ** vip - another JSON object *** net_port_id - uuid *** subnet_id - uuid *** floating_ip_id - uuid *** floating_ip_network_id - uuid ** tenant_id - string ** name - string ** description - string ** enabled - boolean ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR) ** **topology - string enum - (SINGLE, ACTIVE_STANDBY)**

PUT /loadbalancers/{lb_id} * Successful Status Code - 202 * JSON Request Body Attributes ** name - string ** description - string ** enabled - boolean * JSON Response Body Attributes ** id - uuid ** vip - another JSON object * **net_port_id - uuid** * subnet_id - uuid * **floating_ip_id - uuid** * floating_ip_network_id - uuid ** tenant_id - string ** name - string ** description - string ** enabled - boolean ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR) ** **topology - string enum - (SINGLE, ACTIVE_STANDBY)**

GET /loadbalancers/{lb_id} * Successful Status Code - 200 * JSON Response Body Attributes ** id - uuid ** vip - another JSON object * **net_port_id - uuid** * subnet_id - uuid * **floating_ip_id - uuid** * floating_ip_network_id - uuid ** tenant_id - string ** name - string ** description - string ** enabled - boolean ** provisioning_status - string enum - (ACTIVE, PENDING_CREATE, PENDING_UPDATE, PENDING_DELETE, DELETED, ERROR) ** operating_status - string enum - (ONLINE, OFFLINE, DEGRADED, ERROR) ** **topology - string enum - (SINGLE, ACTIVE_STANDBY)**

Security impact

- The VRRP driver must automatically add a security group rule to the amphora's security group to allow VRRP traffic (Protocol number 112) on the same tenant subnet.
- The VRRP driver shall automatically add a security group rule to allow Authentication Header traffic (Protocol number 51).
- VRRP driver shall support authentication-type MD5.
- The HAProxy driver must be updated to automatically add a security group rule that allows multi-peers to synchronize their states.
- Currently HAProxy **does not** support peer authentication, and state sync messages are in plaintext.
- At this point, VRRP shall communicate on the same tenant network. The rationale is to fail-over based on a similar network interfaces condition which the tenant operates experience. Also, VRRP traffic and sync messages shall naturally inherit same protections applied to the tenant network. This may create fake fail-overs if the tenant network is under unplanned, heavy traffic. This is still better than failing over while the master is actually serving tenant's traffic or not failing over at all if the master has failed services. Additionally, the Keepalived shall check the health of the HAproxy service.
- In next steps the following shall be taken into account: * Tenant quotas and supported topologies. * Protection of VRRP Traffic, HAproxy state sync, Router IDs, and pass phrases in both packets and DB.

Notifications impact

None.

Other end user impact

- The operator shall be able to specify the loadbalancer topology in the Octavia configuration file (used by default).

Performance Impact

The Active/Standby can consume up to twice the resources (storage, network, compute) as required by the Single Topology. Nevertheless, one single amphora shall be active (i.e. serving end-user) at any point in time. If the Master amphora is healthy, the backup one shall remain idle until it receives no VRRP advertisements from the master.

The VRRP requires executing health checks in the amphorae at fine grain granularity period. The health checks shall be as lightweight as possible such that VRRP is able to execute all check scripts within a predefined interval. If the check scripts failed to run within this predefined interval, VRRP may become unstable and may alternate the amphorae roles between MASTER and BACKUP incorrectly.

Other deployer impact

- An amphora_topology config option shall be added. The controller worker shall change its taskflow behavior according to the requirement of different topologies.
- By default, the amphora_topology is SINGLE and the ACTIVE/STANDBY topology shall be enabled/requested explicitly by operators.
- The Keepalived version deployed in the amphora image must be newer than 1.2.8 to support unicast VRRP mode.

Developer impact

None.

Implementation

Assignee(s)

Sherif Abdelwahab (abdelwas)

Work Items

- Amphora image update to include Keepalived.
- Data model updates.
- Control Worker extensions.
- Keepalived driver.
- Update Network driver.
- Security rules.
- Update Amphora REST APIs and Jinja Configurations.
- Update Octavia Operator APIs.

Dependencies

Keepalived version deployed in the amphora image must be newer than 1.2.8 to support unicast VRRP mode.

Testing

- Unit tests with tox.
- Function tests with tox.

Documentation Impact

- Description of the different supported topologies: Single, Active/Standby.
- Octavia configuration file changes to enable the Active/Standby topology.
- CLI changes to enable the Active/Standby topology.
- Changes shall be introduced to the amphora APIs: see [11].

References

[1] Implementing High Availability Instances with Neutron using VRRP http://goo.gl/eP71g7

[2] RFC3768 Virtual Router Redundancy Protocol (VRRP)

[3] https://review.opendev.org/#/c/38230/

[4] http://www.keepalived.org/LVS-NAT-Keepalived-HOWTO.html

[5] http://www.formilux.org/archives/haproxy/1003/3259.html

[6] https://blueprints.launchpad.net/octavia/+spec/base-image

[7] https://blueprints.launchpad.net/octavia/+spec/controller-worker

[8] https://blueprints.launchpad.net/octavia/+spec/amphora-driver-interface

[9] https://blueprints.launchpad.net/octavia/+spec/controller

[10] https://blueprints.launchpad.net/octavia/+spec/operator-api

[11] doc/main/api/haproxy-amphora-api.rst

Allow to use Glance image tag to refer to desired Amphora image

https://blueprints.launchpad.net/octavia/+spec/use-glance-tags-to-manage-image

Currently, Octavia allows to define the Glance image ID to be used to boot new Amphoras. This spec suggests another way to define the desired image, by using Glance tagging mechanism.

Problem description

The need to hardcode image ID in the service configuration file has drawbacks.

Specifically, when an updated image is uploaded into Glance, the operator is required to orchestrate configuration file update on all Octavia nodes and then restart all Octavia workers to apply the change. It is both complex and error prone.

Proposed change

The spec suggests an alternative way to configure the desired Glance image to be used for Octavia: using Glance image tagging feature.

Glance allows to tag an image with any tag which is represented by a string value.

With the proposed change, Octavia operator will be able to tell Octavia to use an image with the specified tag. Then Octavia will talk to Glance to determine the exact image ID that is marked with the tag, before booting a new Amphora.

Alternatives

Alternatively, we could make Nova talk to Glance to determine the desired image ID based on the tag provided by Octavia. This approach is not supported by Nova community because they don’t want to impose the complexity into their code base.

Another alternative is to use image name instead of its ID. Nova is capable of fetching the right image from Glance by name as long as the name is unique. This is not optimal in case when the operator does not want to remove the old Amphora image right after a new image is uploaded (for example, if the operator wants to test the new image before cleaning up the old one).

Data model impact

None.

REST API impact

None.

Security impact

Image tags should be managed by the same user that owns the images themselves.

Notifications impact

None.

Other end user impact

The proposed change should not break existing mechanism. To achieve that, the new mechanism will be guarded with a new configuration option that will store the desired Glance tag.

Performance Impact

If the feature is used, Octavia will need to reach to Glance before booting a new Amphora. The performance impact is well isolated and is not expected to be significant.

Other deployer impact

The change couples Octavia with Glance. It should not be an issue since there are no use cases to use Octavia without Glance installed.

The new feature deprecates amp_image_id option. Operators that still use the old image referencing mechanism will be advised to switch to the new option.

Eventually, the old mechanism will be removed from the tree.

Developer impact

None.

Implementation

Assignee(s)

Primary assignee: ihrachys (Ihar Hrachyshka)

Work Items

- introduce glanceclient integration into nova compute driver
- introduce new configuration option to store the glance tag
- introduce devstack plugin support to configure the feature
- provide documentation for the new feature

Dependencies

None.

Testing

Unit tests will be written to cover the feature.

Octavia plugin will be switched to using the new glance image referencing mechanism. Tempest tests will be implemented to test the new feature.

Documentation Impact

New feature should be documented in operator visible guides.

References

4.5.3 Version 0.9 (newton)

Distributor for Active-Active, N+1 Amphorae Setup

Attention: Please review the active-active topology blueprint first (*Active-Active, N+1 Amphorae Setup*)

https://blueprints.launchpad.net/octavia/+spec/active-active-topology

Problem description

This blueprint describes how Octavia implements a *Distributor* to support the *active-active* loadbalancer (LB) solution, as described in the blueprint linked above. It presents the high-level Distributor design and suggests high-level code changes to the current code base to realize this design.

In a nutshell, in an *active-active* topology, an *Amphora Cluster* of two or more active Amphorae collectively provide the loadbalancing service. It is designed as a 2-step loadbalancing process; first, a lightweight *distribution* of VIP traffic over an Amphora Cluster; then, full-featured loadbalancing of traffic over the back-end members. Since a single loadbalancing service, which is addressable by a single VIP address, is served by several Amphorae at the same time, there is a need to distribute incoming requests among these Amphorae -- that is the role of the *Distributor*.

This blueprint uses terminology defined in the Octavia glossary when available, and defines new terms to describe new components and features as necessary.

> **Note:** Items marked with [*P2*] refer to lower priority features to be designed / implemented only after initial release.

Proposed change

- Octavia shall implement a Distributor to support the active-active topology.
- The operator should be able to select and configure the Distributor (e.g., through an Octavia configuration file or [*P2*] through a flavor framework).
- Octavia shall support a pluggable design for the Distributor, allowing different implementations. In particular, the Distributor shall be abstracted through a *driver*, similarly to the current support of Amphora implementations.
- Octavia shall support different provisioning types for the Distributor; including VM-based (the default, similar to current Amphorae), [*P2*] container-based, and [*P2*] external (vendor-specific) hardware.
- The operator shall be able to configure the distribution policies, including affinity and availability (see below for details).

Architecture

High-level Topology Description

- The following diagram illustrates the Distributor's role in an active-active topology:

```
                        Front-End                                       Back-End
Internet                Networks                                        Networks
(world)                 (tenants)                                       (tenants)
                   A       B       C                                     A B C
floating IP
       to VIP f.e. IPs Amphorae b.e.
         LB A                         of      IPs
                                     VIP A Tenant A
  GW
```

(continues on next page)

(continued from previous page)

```
Routerfloating IP
       to VIP f.e. IPs Amphorae b.e.
         LB B                         of     IPs
                                     VIP B Tenant B

      floating IP
       to VIP f.e. IPs Amphorae b.e.
   LB C                         of     IPs
                                       VIP C Tenant C
               arp  arp  arp

             VIPVIPVIP

             IP A  IP B  IP C


                  Distributor
                 (multi-tenant)
```

- In the above diagram, several tenants (A, B, C, ...) share the Distributor, yet the Amphorae, and the front- and back-end (tenant) networks are not shared between tenants. (See also "Distributor Sharing" below.) Note that in the initial code implementing the distributor, the distributor will not be shared between tenants, until tests verifying the security of a shared distributor can be implemented.
- The Distributor acts as a (one-legged) router, listening on each load balancer's VIP and forwarding to one of its Amphorae.
- Each load balancer's VIP is advertised and answered by the Distributor. An `arp` request for any of the VIP addresses is answered by the Distributor, hence any traffic sent for each VIP is received by the Distributor (and forwarded to an appropriate Amphora).
- ARP is disabled on all the Amphorae for the VIP interface.
- The Distributor distributes the traffic of each VIP to an Amphora in the corresponding load balancer Cluster.
- An example of high-level data flow:
  1. Internet clients access a tenant service through an externally visible floating-IP (IPv4 or IPv6).
  2. The GW router maps the floating IP into a loadbalancer's internal VIP on the tenant's front-end network.
  3. (1st packet to VIP only) the GW send an `arp` request on VIP (tenant front-end) network. The Distributor answers the `arp` request with its own MAC address on this network (all the Amphorae on the network can serve the VIP, but do not answer the `arp`).
  4. The GW router forwards the client request to the Distributor.
  5. The Distributor forwards the packet to one of the Amphorae on the tenant's front-end network (distributed according to some policy, as described below), without changing the destination IP (i.e., still using the VIP).

6. The Amphora accepts the packet and continues the flow on the tenant's back-end network as for other Octavia loadbalancer topologies (non active-active).
7. The outgoing response packets from the Amphora are forwarded directly to the GW router (that is, it does not pass through the Distributor).

Affinity of Flows to Amphorae

- Affinity is required to make sure related packets are forwarded to the same Amphora. At minimum, since TCP connections are terminated at the Amphora, all packets that belong to the same flow must be sent to the same Amphora. Enhanced affinity levels can be used to make sure that flows with similar attributes are always sent to the same Amphora; this may be desired to achieve better performance (see discussion below).
- [*P2*] The Distributor shall support different modes of client-to-Amphora affinity. The operator should be able to select and configure the desired affinity level.
- Since the Distributor is L3 and the "heavy lifting" is expected to be done by the Amphorae, this specification proposes implementing two practical affinity alternatives. Other affinity alternatives may be implemented at a later time.

  Source IP and source port In this mode, the Distributor must always send packets from the same combination of Source IP and Source port to the same Amphora. Since the Target IP and Target Port are fixed per Listener, this mode implies that all packets from the same TCP flow are sent to the same Amphora. This is the minimal affinity mode, as without it TCP connections will break.

  Note: related flows (e.g., parallel client calls from the same HTML page) will typically be distributed to different Amphorae; however, these should still be routed to the same back-end. This could be guaranteed by using cookies and/or by synchronizing the stick-tables. Also, the Amphorae in the Cluster could be configured to use the same hashing parameters (avoid any random seed) to ensure all make similar decisions.

  ***Source IP* (default)** In this mode, the Distributor must always send packets from the same source IP to the same Amphora, regardless of port. This mode allows TLS session reuse (e.g., through session ids), where an abbreviated handshake can be used to improve latency and computation time.

  The main disadvantage of sending all traffic from the same source IP to the same Amphora is that it might lead to poor load distribution for large workloads that have the same source IP (e.g., workload behind a single nat or proxy).

  Note on TLS implications: In some (typical) TLS sessions, the additional load incurred for each new session is significantly larger than the load incurred for each new request or connection on the same session; namely, the total load on each Amphora will be more affected by the number of different source IPs it serves than by the number of connections. Moreover, since the total load on the Cluster incurred by all the connections depends on the level of session reuse, spreading a single source IP over multiple Amphorae *increases* the overall load on the Cluster. Thus, a Distributor that uniformly spreads traffic without affinity per source IP (e.g., uses per-flow affinity only) might cause an increase in overall load on the Cluster that is proportional to the number of Amphorae. For example, in a scale-out scenario (where a new Amphora is spawned to share the total load), moving some flows to the new Amphora might increase the overall Cluster load, negating the benefit of scaling-out.

Session reuse helps with the certificate exchange phase. Improvements in performance with the certificate exchange depend on the type of keys used, and is greatest with RSA. Session reuse may be less important with other schemes; shared TLS session tickets are another mechanism that may circumvent the problem; also, upcoming versions of HA-Proxy may be able to obviate this problem by synchronizing TLS state between Amphorae (similar to stick-table protocol).

- Per the agreement at the Mitaka mid-cycle, the default affinity shall be based on source-IP only and a consistent hashing function (see below) shall be used to distribute flows in a predictable manner; however, abstraction will be used to allow other implementations at a later time.

Forwarding with OVS and OpenFlow Rules

- The reference implementation of the Distributor shall use OVS for forwarding and configure the Distributor through OpenFlow rules.
  - OpenFlow rules can be implemented by a software switch (e.g., OVS) that can run on a VM. Thus, can be created and managed by Octavia similarly to creation and management of Amphora VMs.
  - OpenFlow rules are supported by several HW switches, so the same control plane can be used for both SW and HW implementations.
- Outline of Rules
  - A `group` with the `select` method is used to distribute IP traffic over multiple Amphorae. There is one `bucket` per Amphora -- adding an Amphora adds a new `bucket` and deleting and Amphora removes the corresponding `bucket`.
  - The `select` method supports (OpenFlow v1.5) hashed-based selection of the `bucket`. The hash can be set up to use different fields, including by source IP only (default) and by source IP and source port.
  - All buckets route traffic back on the in-port (i.e., no forwarding between ports). This ensures that the same front-end network is used (i.e., the Distributor does not route between front-end networks; therefore, does not mix traffic of different tenants).
  - The `bucket` actions do a re-write of the outgoing packets. It supports re-write of the destination MAC to that of the specific Amphora and re-write of the source MAC to that of the Distributor interface (together these MAC re-writes provide L3 routing functionality).

    Note: alternative re-write rules can be used to support other forwarding mechanisms.
  - OpenFlow rules are also used to answer `arp` requests on the VIP. `arp` requests for each VIP are captured, re-written as `arp` replies with the MAC address of the particular front-end interface and sent back on the in-port. Again, there is no routing between interfaces.
- Handling Amphora failure
  - Initial implementation will assume a fixed size for each cluster (no elasticity). The hashing will be "consistent" by virtue of never changing the number of `buckets`. If the cluster size is changed on the fly (there should not be an API to do so) then there are no guarantees on shuffling.
  - If an Amphora fails then remapping cannot be avoided -- all flows of the failed Amphora must be remapped to a different one. Rather than mapping these flows to other active Amphorae in the cluster, the reference implementation will map all flows to the cluster's *standby* Amphora

(i.e. the "+1" Amphora in this "N+1" cluster). This ensures that the cluster size does not change. The only change in the OpenFlow rules would be to replace the MAC of the failed Amphora with that of the standby Amphora.

  - This implementation is very similar to Active-Standby fail-over. There will be a standby Amphora that can serve traffic in case of failure. The differences from Active-Standby is that a single Amphora acts as a standby for multiple ones; fail-over re-routing is handled through the Distributor (rather than by VRRP); and a whole cluster of Amphorae is active concurrently, to enable support of large workloads.
  - Health Manager will trigger re-creation of a failed Amphora. Once the Amphora is ready it becomes the new *standby* (no changes to OpenFlow rules).
  - [*P2*] Handle concurrent failure of more than a single Amphora

- Handling Distributor failover
  - To handle the event of a Distributor failover caused by a catastrophic failure of a Distributor, and in order to preserve the client to Amphora affinity when the Distributor is replaced, the Amphora registration process with the Distributor should preserve positional information. This should ensure that when a new Distributor is created, Amphorae will be assigned to the same buckets to which they were previously assigned.
  - In the reference implementation, we propose making the Distributor API return the complete list of Amphorae MAC addresses with positional information each time an Amphora is registered or unregistered.

Specific proposed changes

Note: These are changes on top of the changes described in the "Active-Active, N+1 Amphorae Setup" blueprint, (see https://blueprints.launchpad.net/octavia/+spec/active-active-topology)

- Create flow for the creation of an Amphora cluster with N active Amphora and one extra standby Amphora. Set-up the Amphora roles accordingly.
- Support the creation, connection, and configuration of the various networks and interfaces as described in *high-level topology* diagram. The Distributor shall have a separate interface for each loadbalancer and shall not allow any routing between different ports. In particular, when a loadbalancer is created the Distributor should:
  - Attach the Distributor to the loadbalancer's front-end network by adding a VIP port to the Distributor (the LB VIP Neutron port).
  - Configure OpenFlow rules: create a group with the desired cluster size and with the given Amphora MACs; create rules to answer `arp` requests for the VIP address.

  Notes: [*P2*] It is desirable that the Distributor be considered as a router by Neutron (to handle port security, network forwarding without `arp` spoofing, etc.). This may require changes to Neutron and may also mean that Octavia will be a privileged user of Neutron.

  Distributor needs to support IPv6 NDP

  [*P2*] If the Distributor is implemented as a container then hot-plugging a port for each VIP might not be possible.

  If DVR is used then routing rules must be used to forward external traffic to the Distributor rather than rely on `arp`. In particular, DVR messes-up `noarp` settings.

- Support Amphora failure recovery
  - Modify the HM and failure recovery flows to add tasks to notify the ACM when ACTIVE-ACTIVE topology is in use. If an active Amphora fails then it needs to be decommissioned on the Distributor and replaced with the standby.
  - Failed Amphorae should be recreated as a standby (in the new IN_CLUSTER_STANDBY role). The standby Amphora should also be monitored and recovered on failure.
- Distributor driver and Distributor image
  - The Distributor should be supported similarly to an amphora; namely, have its own abstract driver.
  - Distributor image (for reference implementation) should include OVS with a recent version (>1.5) that supports hash-based bucket selection. As is done for Amphorae, Distributor image should be installed with public keys to allow secure configuration by the Octavia controller.
  - Reference implementation shall spawn a new Distributor VM as needed. It shall monitor its health and handle recovery using heartbeats sent to the health monitor in a similar fashion to how this is done presently with Amphorae. [*P2*] Spawn a new Distributor if the number VIPs exceeds a given limit (to limit the number of Neutron ports attached to one Distributor). [*P2*] Add configuration options and/or Operator API to allow operator to request a dedicated Distributor for a VIP (or per tenant).
- Define a REST API for Distributor configuration (no SSH API). See below for details.
- Create data-model for Distributor.

Alternatives

TBD

Data model impact

Add table `distributor` with the following columns:

- **id (`sa.String(36) , nullable=False`)** ID of Distributor instance.
- **compute_id (`sa.String(36), nullable=True`)** ID of compute node running the Distributor.
- **lb_network_ip (`sa.String(64), nullable=True`)** IP of Distributor on management network.
- **status (`sa.String(36), nullable=True`)** Provisioning status
- **vip_port_ids (list of `sa.String(36)`)** List of Neutron port IDs. New VIFs may be plugged into the Distributor when a new LB is created. We may need to store the Neutron port IDs in order to support fail-over from one Distributor instance to another.

Add table `distributor_health` with the following columns:

- **distributor_id (`sa.String(36) , nullable=False`)** ID of Distributor instance.
- **last_update (`sa.DateTime, nullable=False`)** Last time distributor heartbeat was received by a health monitor.

- **busy `(sa.Boolean, nullable=False)`** Field indicating a create / delete or other action is being conducted on the distributor instance (ie. to prevent a race condition when multiple health managers are in use).

Add table `amphora_registration` with the following columns. This describes which Amphorae are registered with which Distributors and in which order:

- **lb_id `(sa.String(36) , nullable=False)`** ID of load balancer.
- **distributor_id `(sa.String(36) , nullable=False)`** ID of Distributor instance.
- **amphora_id `(sa.String(36) , nullable=False)`** ID of Amphora instance.
- **position `(sa.Integer, nullable=True)`** Order in which Amphorae are registered with the Distributor.

REST API impact

Distributor will be running its own rest API server. This API will be secured using two-way SSL authentication, and use certificate rotation in the same way this is done with Amphorae today.

Following API calls will be addressed.

1. Post VIP Plug

   Adding a VIP network interface to the Distributor involves tasks which run outside the Distributor itself. Once these are complete, the Distributor must be configured to use the new interface. This is a REST call, similar to what is currently done for Amphorae when connecting to a new member network.

   lb_id An identifier for the particular loadbalancer/VIP. Used for subsequent register/unregister of Amphorae.

   vip_address The IP of the VIP (for which IP to answer `arp` requests)

   subnet_cidr Netmask for the VIP's subnet.

   gateway Gateway outbound packets from the VIP ip address should use.

   mac_address MAC address of the new interface corresponding to the VIP.

   vrrp_ip In the case of HA Distributor, this contains the IP address that will be used in setting up the allowed address pairs relationship. (See Amphora VIP plugging under the ACTIVE-STANDBY topology for an example of how this is used.)

   host_routes List of routes that should be added when the VIP is plugged.

   alg_extras Extra arguments related to the algorithm that will be used to distribute requests to Amphorae part of this load balancer configuration. This consists of an algorithm name and affinity type. In the initial release of ACTIVE-ACTIVE, the only valid algorithm will be *hash*, and the affinity type may be `Source_IP` or [*P2*] `Source_IP_AND_port`.

2. Pre VIP unplug

   Removing a VIP network interface will involve several tasks on the Distributor to gracefully rollback OVS configuration and other details that were set-up when the VIP was plugged in.

   lb_id ID of the VIP's loadbalancer that will be unplugged.

3. Register Amphorae

   This adds Amphorae to the configuration for a given load balancer. The Distributor should respond with a new list of all Amphorae registered with the Distributor with positional information.

   lb_id ID of the loadbalancer with which the Amphora will be registered

   amphorae List of Amphorae MAC addresses and (optional) position argument in which they should be registered.

4. Unregister Amphorae

   This removes Amphorae from the configuration for a given load balancer. The Distributor should respond with a new list of all Amphorae registered with the Distributor with positional information.

   lb_id ID of the loadbalancer with which the Amphora will be registered

   amphorae List of Amphorae MAC addresses that should be unregistered with the Distributor.

Security impact

The Distributor is designed to be multi-tenant by default. (Note that the first reference implementation will not be multi-tenant until tests can be developed to verify the security of a multi-tenant reference distributor.) Although each tenant has its own front-end network, the Distributor is connected to all, which might allow leaks between these networks. The rationale is two fold: First, the Distributor should be considered as a trusted infrastructure component. Second, all traffic is external traffic before it reaches the Amphora. Note that the GW router has exactly the same attributes; in other words, logically, we can consider the Distributor to be an extension to the GW (or even use the GW HW to implement the Distributor).

This approach might not be considered secure enough for some cases, such as, if LBaaS is used for internal tier-to-tier communication inside a tenant network. Some tenants may want their loadbalancer's VIP to remain private and their front-end network to be isolated. In these cases, in order to accomplish active-active for this tenant we would need separate dedicated Distributor instance(s).

Notifications impact

Other end user impact

Performance Impact

Other deployer impact

Developer impact

Further Discussion

Note: This section captures some background, ideas, concerns, and remarks that were raised by various people. Some of the items here can be considered for future/alternative design and some will hopefully make their way into, yet to be written, related blueprints (e.g., auto-scaled topology).

[P2] Handling changes in Cluster size (manual or auto-scaled)

- The Distributor shall support different mechanism for preserving affinity of flows to Amphorae following a *change in the size* of the Amphorae Cluster.
- The goal is to minimize shuffling of client-to-Amphora mapping during cluster size changes:
  - When an Amphora is removed from the Cluster (e.g., due to failure or scale-down action), all its flows are broken; however, flows to other Amphorae should not be affected. Also, if a drain method is used to empty the Amphora of client flows (in the case of a graceful removal), this should prevent disruption.
  - When an Amphora is *added* to the Cluster (e.g., recovery of a failed Amphora), some new flows should be distributed to the new Amphora; however, most flows should still go to the same Amphora they were distributed to before the new Amphora was added. For example, if the affinity of flows to Amphorae is per Source IP and a new Amphora was just added then the Distributor should forward packets from this IP only one of only two Amphorae: either the same Amphora as before or the Amphora that was added.

  Using a simple hash to maintain affinity does not meet this goal.

  For example, suppose we maintain affinity (for a fixed cluster size) using a hash (for randomizing key distribution) as *chosen_amphora_id = hash(sourceIP # port) mod number_of_amphorae*. When a new Amphora is added or remove the number of Amphorae changes; thus, a different Amphora will be chosen for most flows.
- Below are the couple of ways to tackle this shuffling problem.

  Consistent Hashing Consistent hashing is a hashing mechanism (regardless if key is based on IP or IP/port) that preserves most hash mappings during changes in the size of the Amphorae Cluster. In particular, for a cluster with N Amphorae that grows to N+1 Amphorae, a consistent hashing function ensures that, with high probability, only 1/N of inputs flows will be re-hashed (more precisely, K/N keys will be rehashed). Note that, even with consistent hashing, some flows will be remapped and there is only a statistical bound on the number of remapped flows.

  The "classic" consistent hashing algorithm maps both server IDs and keys to hash values and selects for each key the server with the closest hash value to the key hash value. Lookup generally requires O(log N) to search for the "closest" server. Achieving good distribution requires multiple hashes per server (~10s) -- although these can be pre-computed there is an ~10s*N memory footprint. Other algorithms (e.g., MSFT's Magleb) have better performance, but provide weaker guarantees.

  There are several consistent hashing libraries available. None are supported in OVS.

  - Ketama https://github.com/RJ/ketama
  - Openstack swift https://docs.openstack.org/swift/latest/ring.html#ring
  - Amazon dynamo http://www.allthingsdistributed.com/files/amazon-dynamo-sosp2007.pdf

  We should also strongly consider making any consistent hashing algorithm we develop available to all OpenStack components by making it part of an Oslo library.

  Rendezvous hashing This method provides similar properties to Consistent Hashing (i.e., a hashing function that remaps only 1/N of keys when a cluster with N Amphorae grows to N+1 Amphorae.

For each server ID, the algorithm concatenates the key and server ID and computes a hash. The server with the largest hash is chosen. This approach requires O(N) for each lookup, but is much simpler to implement and has virtually no memory footprint. Through search-tree encoding of the server IDs it is possible to achieve O(log N) lookup, but implementation is harder and distribution is not as good. Another feature is that more than one server can be chosen (e.g., two largest values) to handle larger loads -- not directly useful for the Distributor use case.

Hybrid, Permutation-based approach This is an alternative implementation of consistent hashing that may be simpler to implement. Keys are hashed to a set of buckets; each bucket is pre-mapped to a random permutation of the server IDs. Lookup is by computing a hash of the key to obtain a bucket and then going over the permutation selecting the first server. If a server is marked as "down" the next server in the list is chosen. This approach is similar to Rendezvous hashing if each key is directly pre-mapped to a random permutation (and like it allows more than one server selection). If the number of failed servers is small then lookup is about O(1); memory is O(N * #buckets), where the granularity of distribution is improved by increasing the number of buckets. The permutation-based approach is useful to support clusters of fixed size that need to handle a few nodes going down and then coming back up. If there is an assumption on the number of failures then memory can be reduced to O(max_failures * #buckets). This approach seems to suit the Distributor Active-Active use-case for non-elastic workloads.

- Flow tracking is required, even with the above hash functions, to handle the (relatively few) remapped flows. If an existing flow is remapped, its TCP connection would break. This is acceptable when an Amphora goes down and it flows are mapped to a new one. On the other hand, it may be unacceptable when an Amphora is added to the cluster and 1/N of existing flows are remapped. The Distributor may support different modes, as follows.

  None / Stateless In this mode, the Distributor applies its most recent forwarding rules, regardless of previous state. Some existing flows might be remapped to a different Amphora and would be broken. The client would have to recover and establish a connection with the new Amphora (it would still be mapped to the same back-end, if possible). Combined with consistent (or similar) hashing, this may be good enough for many web applications that are built for failure anyway, and can restore their state upon reconnect.

  Full flow Tracking In this mode, the Distributor tracks existing flows to provide full affinity, i.e., only new flows can be remapped to different Amphorae. The Linux connection tracking may be used (e.g., through IPTables or through OpenFlow); however, this might not scale well. Alternatively, the Distributor can use an independent mechanism similar to HA-Proxy sticky-tables to track the flows. Note that the Distributor only needs to track the mapping per source IP and source port (unlike Linux connection tracking which follows the TCP state and related connections).

  Use Ryu Ryu is a well supported and tested python binding for issuing OpenFlow commands. Especially since Neutron recently moved to using this for many of the things it does, using this in the Distributor might make sense for Octavia as well.

Forwarding Data-path Implementation Alternatives

The current design uses L2 forwarding based only on L3 parameters and uses Direct Return routing (one-legged). The rational behind this approach is to keep the Distributor as light as possible and have the Amphorae do the bulk of the work. This allows one (or a few) Distributor instance(s) to serve all traffic even for very large workloads. Other approaches are possible.

2-legged Router

- Distributor acts as router, being in-path on both directions.
- New network between Distributor and Amphorae -- Only Distributor on VIP subnet.
- No need to use MAC forwarding -- use routing rules

LVS

Use LVS for Distributor.

DNS

Use DNS for the Distributor.

- Use DNS to map to particular Amphorae. Distribution will be of domain name rather than VIP.
- No problem with per-flow affinity, as client will use same IP for entire TCP connection.
- Need a different public IP for each Amphora (no VIP)

Pure SDN

- Implement the OpenFlow rules directly in the network, without a Distributor instance.
- If the network infrastructure supports this then the Distributor can become more robust and very lightweight, making it practical to have a dedicated Distributor per VIP (only the rules will be dedicated as the network and SDN controller are shared resources)

Distributor Sharing

- The initial implementation of the Distributor will not be shared between tenants until tests can be written to verify the security of this solution.
- The implementation should support different Distributor sharing and cardinality configurations. This includes single-shared Distributor, multiple-dedicated Distributors, and multiple-shared Distributors. In particular, an abstraction layer should be used and the data-model should include an association between the load balancer and Distributor.
- A shared Distributor uses the least amount of resources, but may not meet isolation requirements (performance and/or security) or might become a bottleneck.

Distributor High-Availability

- The Distributor should be highly-available (as this is one of the motivations for the active-active topology). Once the initial active-active functionality is delivered, developing a highly available distributor should take a high priority.
- A mechanism similar to the VRRP used on ACTIVE-STANDBY topology Amphorae can be used.
- Since the Distributor is stateless (for fixed cluster sizes and if no connection tracking is used) it is possible to set up an active-active configuration and advertise more than one Distributor (e.g, for ECMP).
- As a first step, the initial implementation will use a single Distributor instance (i.e., will not be highly-available). Health Manager will monitor the Distributor health and initiate recovery if needed.
- The implementation should support plugging-in a hardware-based implementation of the Distributor that may have its own high-availability support.
- In order to preserve client to Amphora affinity in the case of a failover, a VRRP-like HA Distributor has several options. We could potentially push Amphora registrations to the standby Distributor with the position arguments specified, in order to guarantee the active and standby Distributor always have the same configuration. Or, we could invent and utilize a synchronization protocol between the active and standby Distributors. This will be explored and decided when an HA Distributor specification is written and approved.

Implementation

Assignee(s)

Work Items

Dependencies

Testing

- Unit tests with tox.
- Function tests with tox.

Documentation Impact

References

https://blueprints.launchpad.net/octavia/+spec/base-image https://blueprints.launchpad.net/octavia/+spec/controller-worker https://blueprints.launchpad.net/octavia/+spec/amphora-driver-interface https://blueprints.launchpad.net/octavia/+spec/controller https://blueprints.launchpad.net/octavia/+spec/operator-api *Octavia HAProxy Amphora API* https://blueprints.launchpad.net/octavia/+spec/active-active-topology

Active-Active, N+1 Amphorae Setup

https://blueprints.launchpad.net/octavia/+spec/active-active-topology

Problem description

This blueprint describes how Octavia implements an *active-active* loadbalancer (LB) solution that is highly-available through redundant Amphorae. It presents the high-level service topology and suggests high-level code changes to the current code base to realize this scenario. In a nutshell, an *Amphora Cluster* of two or more active Amphorae collectively provide the loadbalancing service.

The Amphora Cluster shall be managed by an *Amphora Cluster Manager* (ACM). The ACM shall provide an abstraction that allows different types of active-active features (e.g., failure recovery, elasticity, etc.). The initial implementation shall not rely on external services, but the abstraction shall allow for interaction with external ACMs (to be developed later).

This blueprint uses terminology defined in Octavia glossary when available, and defines new terms to describe new components and features as necessary.

> **Note:** Items marked with [*P2*] refer to lower priority features to be designed / implemented only after initial release.

Proposed change

A tenant should be able to start a highly-available, loadbalancer for the tenant's backend services as follows:

- The operator should be able to configure an active-active topology through an Octavia configuration file or [*P2*] through a Neutron flavor, which the loadbalancer shall support. Octavia shall support active-active topologies in addition to the topologies that it currently supports.
- In an active-active topology, a cluster of two or more amphorae shall host a replicated configuration of the load-balancing services. Octavia will manage this *Amphora Cluster* as a highly-available service using a pool of active resources.
- The Amphora Cluster shall provide the load-balancing services and support the configurations that are supported by a single Amphora topology, including L7 load-balancing, SSL termination, etc.
- The active-active topology shall support various Amphora types and implementations; including, virtual machines, [*P2*] containers, and bare-metal servers.
- The operator should be able to configure the high-availability requirements for the active-active load-balancing services. The operator shall be able to specify the number of healthy Amphorae that must exist in the load-balancing Amphora Cluster. If the number of healthy Amphorae drops under the desired number, Octavia shall automatically and seamlessly create and configure a new Amphora and add it to the Amphora Cluster. [*P2*] The operator should be further able to define that the Amphora Cluster shall be allocated on separate physical resources.
- An Amphora Cluster will collectively act to serve as a single logical loadbalancer as defined in the Octavia glossary. Octavia will seamlessly distribute incoming external traffic among the Amphorae in the Amphora Cluster. To that end, Octavia will employ a *Distributor* component that will forward external traffic towards the managed amphora instances. Conceptually, the Distributor provides an extra level of load-balancing for an active-active Octavia application, albeit a simplified one. Octavia should be able to support several Distributor implementations (e.g., software-based

and hardware-based) and different affinity models (at minimum, flow-affinity should be supported to allow TCP connectivity between clients and Amphorae).

- The detailed design of the Distributor component will be described in a separate document (see "Distributor for Active-Active, N+1 Amphorae Setup", active-active-distributor.rst).

High-level Topology Description

Single Tenant

- The following diagram illustrates the active-active topology:

```
                   Front-End                  Back-End
Internet            Network                   Network
(world)             (tenant)                  (tenant)

 floating IP
Router  to LB VIP           Tenant
  GW   IP  Amphora  IP Service
                    (1)   back      (1)
                          VIP
                            MGMT
    IP             Tenant
                       Service
      Distri-   IP                             (2)
       butor
     VIP   IP  Amphora  IP
        MGMT        (2)   back     Tenant
    IP    arp   VIP  Service
                   MGMT             (3)
    IP

     Octavia LBaaS
    Controller
      IP  Amphora  IP
        Amphora           (k)   back
      Cluster Mgr.      VIP     Tenant
             MGMT         Service
    IP               (m)


Management                    Amphora Cluster    Back-end Pool
Network                           1..k                1..m
```

- An example of high-level data-flow:
  1. Internet clients access a tenant service through an externally visible floating-IP (IPv4 or IPv6).
  2. If IPv4, a gateway router maps the floating IP into a loadbalancer's internal VIP on the tenant's front-end network.

3. The (multi-tenant) Distributor receives incoming requests to the loadbalancer's VIP. It acts as a one-legged direct return LB, answering `arp` requests for the loadbalancer's VIP (see Distributor spec.).
4. The Distributor distributes incoming connections over the tenant's Amphora Cluster, by forwarding each new connection opened with a loadbalancer's VIP to a front-end MAC address of an Amphora in the Amphora Cluster (layer-2 forwarding). *Note*: the Distributor may implement other forwarding schemes to support more complex routing mechanisms, such as DVR (see Distributor spec.).
5. An Amphora receives the connection and accepts traffic addressed to the loadbalancer's VIP. The front-end IPs of the Amphorae are allocated on the tenant's front-end network. Each Amphora accepts VIP traffic, but does not answer `arp` request for the VIP address.
6. The Amphora load-balances the incoming connections to the back-end pool of tenant servers, by forwarding each external request to a member on the tenant network. The Amphora also performs SSL termination if configured.
7. Outgoing traffic traverses from the back-end pool members, through the Amphora and directly to the gateway (i.e., not through the Distributor).

Multi-tenant Support

- The following diagram illustrates the active-active topology with multiple tenants:

```
                    Front-End                   Back-End
Internet             Networks                   Networks
(world)              (tenant)                   (tenant)
                      B  A                         A
        floating IP
 to LB VIP A           Tenant A
RouterA IP Amphora A IPService
  GW         (1)   back      (1)
 floating IP        VIP
         to LB VIP B          MGMT
    IP           Tenant A
                                         Service
  M                    B  A        (2)
                        A IP Amphora A IP
                             (2)   back
                           VIP    Tenant A
                             MGMT         Service
    IP              (3)

                      B  A                      B


              IP AA IP Amphora A IP
                      (k)   back    Tenant A
      Distri- VIParp   VIP  Service
       butor            MGMT              (m)
               IP

```

(continues on next page)

(continued from previous page)

```
              IP B                               tenant A
                 = = = = = = = = = = = = = = = = = = = = =
              VIP         B   tenant B
       B IP Amphora B IP
        MGMT   arp        (1)   back    Tenant B
    IP                VIP  Service
                      MGMT             (1)
    IP

  M   Octavia LBaaS  B                          B
    Controller
     B IP Amphora B IP
        Amphora              (q)   back
      Cluster Mgr.         VIP    Tenant B
                MGMT         Service
    IP                 (r)


Management                      Amphora Clusters      Back-end Pool
Network                         A(1..k), B(1..q)    A(1..m),B(1..r)
```

- Both tenants A and B share the Distributor, but each has a different front-end network. The Distributor listens on both loadbalancers' VIPs and forwards to either A's or B's Amphorae.
- The Amphorae and the back-end (tenant) networks are not shared between tenants.

Problem Details

- Octavia should support different Distributor implementations, similar to its support for different Amphora types. The operator should be able to configure different types of algorithms for the Distributor. All algorithms should provide flow-affinity to allow TLS termination at the amphora. See *Distributor for Active-Active, N+1 Amphorae Setup* for details.
- Octavia controller shall seamlessly configure any newly created Amphora ([*P2*] including peer state synchronization, such as sticky-tables, if needed) and shall reconfigure the other solution components (e.g., Neutron) as needed. The controller shall further manage all Amphora life-cycle events.
- Since it is impractical at scale for peer state synchronization to occur between all Amphorae part of a single load balancer, Amphorae that are all part of a single load balancer configuration need to be divided into smaller peer groups (consisting of 2 or 3 Amphorae) with which they should synchronize state information.

Required changes

The active-active loadbalancers require the following high-level changes:

Amphora related changes

- Updated Amphora image to support active-active topology. The front-end still has both a unique IP (to allow direct addressing on front-end network) and a VIP; however, it should not answer ARP requests for the VIP address (all Amphorae in a single Amphora Cluster concurrently serve the same VIP). Amphorae should continue to have a management IP on the LB Network so Octavia can configure them. Amphorae should also generally support hot-plugging interfaces into back-end tenant networks as they do in the current implementation. [*P2*] Finally, the Amphora configuration may need to be changed to randomize the member list, in order to prevent synchronized decisions by all Amphorae in the Amphora Cluster.
- Extend data model to support active-active Amphora. This is somewhat similar to active-passive (VRRP) support. Each Amphora needs to store its IP and port on its front-end network (similar to ha_ip and ha_port_id in the current model) and its role should indicate it is in a cluster.

  The provisioning status should be interpreted as referring to an Amphora only and not the load-balancing service. The status of the load balancer should correspond to the number of `ONLINE` Amphorae in the Cluster. If all Amphoae are `ONLINE`, the load balancer is also `ONLINE`. If a small number of Amphorae are not `ONLINE`, then the load balancer is `DEGRADED`. If enough Amphorae are not `ONLINE` (past a threshold), then the load balancer is `DOWN`.
- Rework some of the controller worker flows to support creation and deletion of Amphorae by the ACM in an asynchronous manner. The compute node may be created/deleted independently of the corresponding Amphora flow, triggered as events by the ACM logic (e.g., node update). The flows do not need much change (beyond those implied by the changes in the data model), since the post-creation/pre-deletion configuration of each Amphora is unchanged. This is also similar to the failure recovery flow, where a recovery flow is triggered asynchronously.
- Create a flow (or task) for the controller worker for (de-)registration of Amphorae with Distributor. The Distributor has to be aware of the current `ONLINE` Amphorae, to which it can forward traffic. [*P2*] The Distributor can do very basic monitoring of the Amphorae health (primarily to make sure network connectivity between the Distributor and Amphorae is working). Monitoring pool member health will remain the purview of the pool health monitors.
- All the Amphorae in the Amphora Cluster shall replicate the same listeners, pools, and TLS configuration, as they do now. We assume all Amphorae in the Amphora Cluster can perform exactly the same load-balancing decisions and can be treated as equivalent by the Distributor (except for affinity considerations).
- Extend the Amphora (REST) API and/or *Plug VIP* task to allow disabling of `arp` on the VIP.
- In order to prevent losing session_persistence data in the event of an Amphora failure, the Amphorae will need to be configured to share session_persistence data (via stick tables) with a subset of other Amphorae that are part of the same load balancer configuration (ie. a peer group).

Amphora Cluster Manager driver for the active-active topology (*new*)

- Add an active-active topology to the topology types.
- Add a new driver to support creation/deletion of an Amphora Cluster via an ACM. This will re-use existing controller-worker flows as much as possible. The reference ACM will call the existing drivers to create compute nodes for the Amphorae and configure them.
- The ACM shall orchestrate creation and deletion of Amphora instances to meet the availability requirements. Amphora failover will utilize the existing health monitor flows, with hooks to notify the ACM when ACTIVE-ACTIVE topology is used. [*P2*] ACM shall handle graceful amphora removal via draining (delay actual removal until existing connections are terminated or some timeout has passed).
- Change the flow of LB creation. The ACM driver shall create an Amphora Cluster instance for each new loadbalancer. It should maintain the desired number of Amphorae in the Cluster and meet the high-availability configuration given by the operator. *Note*: a base functionality is already supported by the Health Manager; it may be enough to support a fixed or dynamic cluster size. In any case, existing flows to manage Amphora life cycle will be re-used in the reference ACM driver.
- The ACM shall be responsible for providing health, performance, and life-cycle management at the Cluster-level rather than at Amphora-level. Maintaining the loadbalancer status (as described above) by some function of the collective status of all Amphorae in the Cluster is one example. Other examples include tracking configuration changes, providing Cluster statistics, monitoring and maintaining compute nodes for the Cluster, etc. The ACM abstraction would also support pluggable ACM implementations that may provide more advance capabilities (e.g., elasticity, AZ aware availability, etc.). The reference ACM driver will re-use existing components and/or code which currently handle health, life-cycle, etc. management for other load balancer topologies.
- New data model for an Amphora Cluster which has a one-to-one mapping with the loadbalancer. This defines the common properties of the Amphora Cluster (e.g., id, min. size, desired size, etc.) and additional properties for the specific implementation.
- Add configuration file options to support configuration of an active-active Amphora Cluster. Add default configuration. [*P2*] Add Operator API.
- Add or update documentation for new components added and new or changed functionality.
- Communication between the ACM and Distributors should be secured using two-way SSL certificate authentication much the same way this is accomplished between other Octavia controller components and Amphorae today.

Network driver changes

- Support the creation, connection, and configuration of the various networks and interfaces as described in 'high-level topology' diagram.
- Adding a new loadbalancer requires attaching the Distributor to the loadbalancer's front-end network, adding a VIP port to the Distributor, and configuring the Distributor to answer `arp` requests for the VIP. The Distributor shall have a separate interface for each loadbalancer and shall not allow any routing between different ports; in particular, Amphorae of different tenants must not be able to communicate with each other. In the reference implementation, this will be accomplished by using separate OVS bridges per load balancer.

- Adding a new Amphora requires attaching it to the front-end and back-end networks (similar to current implementation), adding the VIP (but with `arp` disabled), and registering the Amphora with the Distributor. The tenant's front-end and back-end networks must allow attachment of dynamically created Amphorae by involving the ACM (e.g., when the health monitor replaces a failed Amphora). ([*P2*] extend the LBaaS API to allow specifying an address range for new Amphorae usage, e.g., a subnet pool).

Amphora health-monitoring support

- Modify Health Manager to manage the health for an Amphora Cluster through the ACM; namely, forward Amphora health change events to the ACM, so it can decide when the Amphora Cluster is considered to be in healthy state. This should be done in addition to managing the health of each Amphora. [*P2*] Monitor the Amphorae also on their front-end network (i.e., from the Distributor).

Distributor support

- **Note:** as mentioned above, the detailed design of the Distributor component is described in a separate document). Some design considerations are highlighted below.
- The Distributor should be supported similarly to an Amphora; namely, have its own abstract driver.
- For a reference implementation, add support for a Distributor image.
- Define a REST API for Distributor configuration (no SSH API). The API shall support:
  - Add and remove a VIP (loadbalancer) and specify distribution parameters (e.g., affinity, algorithm, etc.).
  - Registration and de-registration of Amphorae.
  - Status
  - [*P2*] Macro-level stats
- Spawn Distributors (if using on demand Distributor compute nodes) and/or attach to existing ones as needed. Manage health and life-cycle of the Distributor(s). Create, connect, and configure Distributor networks as necessary.
- Create data model for the Distributor.
- Add Distributor driver and flows to (re-)configure the Distributor on creation/destruction of a new loadbalancer (add/remove loadbalancer VIP) and [*P2*] configure the distribution algorithm for the loadbalancer's Amphora Cluster.
- Add flows to Octavia to (re-)configure the Distributor on adding/removing Amphorae from the Amphora Cluster.

Packaging

- Extend Octavia installation scripts to create an image for the Distributor.

Alternatives

- **Use external services to manage the cluster directly.** This utilizes functionality that already exists in OpenStack (e.g., like Heat and Ceilometer) rather than replicating it. This approach would also benefit from future extensions to these services. On the other hand, this adds undesirable dependencies on other projects (and their corresponding teams), complicates handling of failures, and require defensive coding around service calls. Furthermore, these services cannot handle the LB-specific control configuration.
- **Implement a nested Octavia** Use another layer of Octavia to distribute traffic across the Amphora Cluster (i.e., the Amphorae in the Cluster are back-end members of another Octavia instance). This approach has the potential to provide greater flexibility (e.g., provide NAT and/or more complex distribution algorithms). It also potentially reuses existing code. However, we do not want the Distributor to proxy connections so HA-Proxy cannot be used. Furthermore, this approach might significantly increase the overhead of the solution.

Data model impact

- loadbalancer table
    - *cluster_id*: associated Amphora Cluster (no changes to table, 1-1 relationship from Cluster data-model)
- lb_topology table
    - new value: `ACTIVE_ACTIVE`
- amphora_role table
    - new value: `IN_CLUSTER`
- Distributor table (*new*): Distributor information, similar to Amphora. See *Distributor for Active-Active, N+1 Amphorae Setup*
- Cluster table (*new*): an extension to loadbalancer (i.e., one-to-one mapping to load-balancer)
    - *id* (primary key)
    - *cluster_name*: identifier of Cluster instance for Amphora Cluster Manager
    - *desired_size*: required number of Amphorae in Cluster. Octavia will create this many active-active Amphorae in the Amphora Cluster.
    - *min_size*: number of `ACTIVE` Amphorae in Cluster must be above this number for Amphora Cluster status to be `ACTIVE`
    - *cooldown*: cooldown period between successive add/remove Amphora operations (to avoid thrashing)
    - *load_balancer_id*: 1:1 relationship to loadbalancer
    - *distributor_id*: N:1 relationship to Distributor. Support multiple Distributors

  - *provisioning_status*
  - *operating_status*
  - *enabled*
  - *cluster_type*: type of Amphora Cluster implementation

REST API impact

- Distributor REST API -- This is a new internal API that will be secured via two-way SSL certificate authentication. See *Distributor for Active-Active, N+1 Amphorae Setup*
- Amphora REST API -- support configuration of disabling `arp` on VIP.
- [*P2*] LBaaS API -- support configuration of desired availability, perhaps by selecting a flavor (e.g., gold is a minimum of 4 Amphorae, platinum is a minimum of 10 Amphora).
- Operator API --
  - Topology to use
  - Cluster type
  - Default availability parameters for the Amphora Cluster

Security impact

- See *Distributor for Active-Active, N+1 Amphorae Setup* for Distributor related security impact.

Notifications impact

None.

Other end user impact

None.

Performance Impact

ACTIVE-ACTIVE should be able to deliver significantly higher performance than SINGLE or ACTIVE-STANDBY topology. It will consume more resources to deliver this higher performance.

Other deployer impact

The reference ACM becomes a new process that is part of the Octavia control components (like the controller worker, health monitor and housekeeper). If the reference implementation is used, a new Distributor image will need to be created and stored in glance much the same way the Amphora image is created and stored today.

Developer impact

None.

Implementation

Assignee(s)

@TODO

Work Items

@TODO

Dependencies

@TODO

Testing

- Unit tests with tox.
- Function tests with tox.
- Scenario tests.

Documentation Impact

Need to document all new APIs and API changes, new ACTIVE-ACTIVE topology design and features, and new instructions for operators seeking to deploy Octavia with ACTIVE-ACTIVE topology.

References

https://blueprints.launchpad.net/octavia/+spec/base-image https://blueprints.launchpad.net/octavia/+spec/controller-worker https://blueprints.launchpad.net/octavia/+spec/amphora-driver-interface https://blueprints.launchpad.net/octavia/+spec/controller https://blueprints.launchpad.net/octavia/+spec/operator-api *Octavia HAProxy Amphora API* https://blueprints.launchpad.net/octavia/+spec/active-active-topology

Add statistics gathering API for loadbalancer

https://blueprints.launchpad.net/octavia/+spec/stats-support

Problem description

Currently, Octavia does not support the gathering of loadbalancer statistics. This causes inconsistencies between the Octavia and Neutron-LBaaS APIs. Another point is that the statistics data we get from the Octavia API for the listener only reflects the first record for the listener in the Octavia database, since we're supporting more topologies than SINGLE, this needs to be to fixed too.

Proposed change

Add one more data 'request_errors' to indicate the number of request errors for each listener, we can get this data from the stats of haproxy 'ereq'.

Add a new module 'stats' to octavia.common with a class 'StatsMixin' to do the actual statistics calculation for both listener and loadbalancer. Make the mixin class as a new base class for octavia.api.v1.controllers.listener_statistics.ListenerStatisticsController, to make sure we get correct stats from Octavia API.

Add a new module 'loadbalancer_statistics' to octavia.api.v1.controllers with a class LoadbalancerStatisticsController to provide a new REST API for gathering statistics at the loadbalancer level.

Use evenstream to serialize the statistics messages from the octavia to neutron-lbaas via oslo_messaging, to keep consistent with neutron-lbaas API.

Alternatives

Update the 'stats' method in neutron-lbaas for octavia driver, allow the neutron-lbaas to get stats from octavia through REST API request, to keep consistent with neutron-lbaas API.

Data model impact

One new column for table listener_statistics will be introduced to represent request errors:

Field	Type	Null	Key	Default	Extra
request_errors	bigint(20)	NO		NULL	

REST API impact

Add 'request_errors' in the response of list listener statistics:

Example List listener statistics: JSON response

```
{
    "listener": {
        "bytes_in": 0,
        "bytes_out": 0,
        "active_connections": 0,
        "total_connections": 0,
        "request_errors": 0
    }
}
```

Add a new API to list loadbalancer statistics

Lists loadbalancer statistics.

Request Type	GET	
Endpoint	`URL/v1/loadbalancers/{lb_id}/stats`	
Response Codes	Success	200
	Error	401, 404, 500

Example List loadbalancer statistics: JSON response

```
{
    "loadbalancer": {
        "bytes_in": 0,
        "bytes_out": 0,
        "active_connections": 0,
        "total_connections": 0,
        "request_errors": 0,
        "listeners": [{
            "id": "uuid"
            "bytes_in": 0,
            "bytes_out": 0,
            "active_connections": 0,
            "total_connections": 0,
```

(continues on next page)

(continued from previous page)

```
            "request_errors": 0,
        }]
    }
}
```

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

None

Implementation

Assignee(s)

li, chen <shchenli@cn.ibm.com>

Work Items

- Extend current stats collection for listener amphora
- Add module 'stats'
- Add new API for gathering statistics at the loadbalancer level
- Update stats to neutron database

Dependencies

None

Testing

Function tests with tox.

Documentation Impact

Changes shall be introduced to the octavia APIs: see [1]

References

[1] https://docs.openstack.org/api-ref/load-balancer/v1/octaviaapi.html

4.5.4 Version 1.0 (pike)

Provider Flavor Framework

https://blueprints.launchpad.net/octavia/+spec/octavia-lbaas-flavors

A Provider Flavor framework provides a mechanism for providers to specify capabilities that are not currently handled via the octavia api. It allows the operators to enable capabilities that may possibly be unique to a particular provider or simply just not available at the moment within octavia. If it is a common feature it is highly encouraged to have the non-existing features implemented via the standard Octavia api. In addition operators can configure different flavors from a maintained list of provider capabilities. This framework enables providers to supply new features with speed to market and provides operators with an ease of use experience.

Problem description

Flavors are used in various services for specifying service capabilities and other parameters. Having the ability to create loadbalancers with various capabilities (such as HA, throughput or ddos protection) gives users a way to better plan their LB services and get a benefit of LBaaS functions which are not a part of Octavia API. Since Octavia will become the new OpenStack LBaaS API, a new flavors API should be developed inside Octavia.

As for now, Octavia does not support multi providers. The ability to define different LBaaS providers is a mandatory feature for Octavia to be Openstack LbaaS API. Therefore, this spec depends on adding multi providers support to Octavia. Service providers will be configured via Octavia configuration file.

Its important to mention that adding flavors capability to Octavia is not actually dependent on the work for LBaaS API spinout, from Neutron to Octavia, to be completed. This capability can be added to Octavia but not actually used until the API spinout is complete and Octavia becomes the official OpenStack LBaaS API.

This spec is based on two existing specs from neutron:

Service Flavor Framework Flavor framework - Templates and meta-data

However, this is a spec for the first and basic flavors support. Following capabilities are not part of this spec:

- Providing parameterized metainfo templates for provider profiles.
- Providing meta data for specific LBaaS object as part of its creation.

Proposed change

The Provider Flavor framework enables the ability to create distinct provider flavor profiles of supported parameters. Operators will have the ability to query the provider driver interface for a list of supported parameters. Operators can view the said list by provider and create flavors by selecting one or many parameters from the list. The parameters that will be used to enable specific functionality will be json type in transit and at rest. This json payload is assigned to a provider and a flavor name. Users then have the option of selecting from any of the existing flavors and submitting the selected flavor upon the creation of the load balancer. The following flavor name examples can be, but not limited to dev, stage, prod or bronze, silver, gold. A provider can have many flavor names and a flavor name can be used by only one provider. Each provider/flavor pair is assigned a group of meta-parameters and forms a flavor profile. The flavor name or id is submitted when creating a load balancer.

The proposal is to add LBaaS service flavoring to Octavia. This will include following aspects:

- Adding new flavors API to Octavia API
- Adding flavors models to Octavia
- Adding flavors db tables to Octavia database
- Adding DB migration for new DB objects
- Ensuring backwards compatibility for loadbalancer objects which were created before flavors support. This is for both cases, when loadbalancer was created before multi providers support and when loadbalancer was created with certain provider.
- Adding default entries to DB tables representing the default Octavia flavor and default Octavia provider profile.

- Adding "default" flavor to devstack plugin.

A sample use case of the operator flavor workflow would be the following:

- The operator queries the provider capabilities
- The operator create flavor profile
- The flavor profile is validated with provider driver
- The flavor profile is stored in octavia db
- The end user creates lb with the flavor
- The profile is validated against driver once again, upon every lb-create

Alternatives

An alternative is patchset-5 within this very same spec. While the concept is the same, the design is different. Differences with patchset-5 to note is primarily with the data schemas. With patchset-5 the metadata that is passed to the load balancer has a one to one relationship with the provider. Also key/values pairs are stored in json as opposed to in normalized tables. And a list of provider supported capabilities is not maintained. That said this alternative design is an option.

Data model impact

DB table 'flavor_profile' introduced to represent the profile that is created when combining a provider with a flavor.

Field	Type	Null	Key	Default
id	varchar(36)	NO	PK	generated
provider_name	varchar(255)	NO		
metadata	varchar(4096)	NO		

Note: The provider_name is the name the driver is advertised as via setuptools entry points. This will be validated when the operator uploads the flavor profile and the metadata is validated.

DB table 'flavor' introduced to represent flavors.

Field	Type	Null	Key	Default
id	varchar(36)	NO	PK	generated
name	varchar(255)	NO	UK	
description	varchar(255)	YES		NULL
enabled	tinyint(1)	NO		True
flavor_profile_id	varchar(36)	NO	FK	

DB table attribute 'load_balancer.flavor_id' introduced to link a flavor to a load_balancer.

Field	Type	Null	Key	Default
flavor_id	varchar(36)	YES	FK1	NULL

REST API impact

FLAVOR(/flavors)

Attribute Name	Type	Access	Default Value	Validation/ Conversion	Description
id	string (UUID)	RO, admin	generated	N/A	identity
name	string	RO, admin	”	string	human-readable name
description	string	RO, admin	”	string	human-readable description
enabled	bool	RO, admin	true	bool	toggle
flavor_profile_id	string	RO, admin		string	human-readable flavor_profile_id

FLAVOR PROFILE(/flavorprofiles)

Attribute Name	Type	Access	Default Value	Validation/ Conversion	Description
id	string (UUID)	admin	generated	N/A	identity
name	string	admin	”	string	human-readable name
provider-id	string	admin	”	string	human-readable provider-id
metadata	string	admin	{}	json	flavor meta parameters

Security impact

The policy.json will be updated to allow all users to query the flavor listing and request details about a specific flavor entry, with the exception of flavor metadata. All other REST points for create/update/delete operations will be admin only. Additionally, the CRUD operations for Provider Profiles will be restricted to administrators.

Notifications impact

N/A

Other end user impact

An existing LB cannot be updated with a different flavor profile. A flavor profile can only be applied upon the creation of the LB. The flavor profile will be immutable.

Performance Impact

There will be a minimal overhead incurred when the logical representation is scheduled onto the actual backend. Once the backend is selected, direct communications will occur via driver calls.

IPv6 impact

None

Other deployer impact

The deployer will need to craft flavor configurations that they wish to expose to their users. During migration the existing provider configurations will be converted into basic flavor types. Once migrated, the deployer will have the opportunity to modify the flavor definitions.

Developer impact

The expected developer impact should be minimal as the framework only impacts the initial scheduling of the logical service onto a backend. The driver implementations should remain unchanged except for the addition of the metainfo call.

Community impact

This proposal allows operators to offer services beyond those directly implemented, and to do so in a way that does not increase community maintenance or burden.

Provider driver impact

The provider driver should have the following abilities:

- Provide an interface to describe the available supported metadata options
- Provide an interface to validate the flavor metadata
- Be able to accept the flavor metadata parameters
- Exception handling for non-supported metadata

Implementation

Assignee(s)

- Evgeny Fedoruk (evgenyf)
- Carlos Puga (cpuga)

Work Items

- Implement the new models
- Implement the REST API Extension (including tests)
- Implementation migration script for existing deployments.
- Add client API support
- Add policies to the Octavia RBAC system

Dependencies

Depends on provider support and provider drivers that support the validation interface and accept the flavor profile metadata.

Testing

Tempest Tests

Tempest testing including new API and scenario tests to validate new entities.

Functional Tests

Functional tests will need to be created to cover the API and database changes.

API Tests

The new API extensions will be tested using functional tests.

Documentation Impact

User Documentation

User documentation will need be included to describe to users how to use flavors when building their logical topology.

Operator Documentation

Operator documentation will need to be created to detail how to manage Flavors, Providers and their respective Profiles.

Developer Documentation

Provider driver implementation documentation will need to be updated to cover the new interfaces expected of provider drivers and the structure of the metadata provided to the driver.

API Reference

The API reference documentation will need to be updated for the new API extensions.

References

[1] https://docs.openstack.org/api-ref/load-balancer/v2/index.html

LBaaS Alternative Monitoring IP/Port

https://blueprints.launchpad.net/octavia/+spec/lbaas-health-monitoring-port

In the current state, the health monitor IP address/port pair is derived from a load balancer's pool member's address and protocol port. In some use cases it would be desirable to monitor a different IP address/port pair for the health of a load balanced pool's member than the already specified address and protocol port. Due to the current state this is not possible.

Problem description

The use case where this would be desirable would be when the End User is making the health monitor application on the member available on a IP/port that is mutually exclusive to the IP/port of the application that is being load balanced on the member. The End User would find this advantageous when attempting to limit access to health diagnostic information by not allowing it to be served over the main ingress IP/port of their application.

Beyond limiting access to any health APIs, it allows the End Users to design different methods of health monitoring, such as creating distinct daemons responsible for the health of their hosts applications.

Proposed change

The creation of a pool member would now allow the specification of an IP address and port to monitor health. The process used to assess the health of pool members would now use this new IP address and port to diagnose the member.

If a health monitor IP address or port is not specified the default behavior would be to use the IP address and port specified by the member.

There would likely need to be some Horizon changes to support this feature, however by maintaining the old behavior as the default we will not create a strong dependency.

Alternatives

An alternative is to not allow this functionality, and force all End Users to ensure their health checks are available over the member's load balanced IP address and protocol port.

As stated in the *Problem Description* this would force End Users to provide additional security around their health diagnostic information so that they do not expose it to unintended audiences. Pushing this requirement on the End User is a heavier burden and limits their configuration options of the applications they run on Openstack that are load balanced.

Data model impact

The Member data model would gain two new member fields called monitor_port and monitor_address. These two member fields would store the port and IP address, respectively, that the monitor will query for the health of the load balancer's listener's pool member.

It is important to have the default behavior fall back on the address and protocol port of the member as this will allow any migrations to not break existing deployments of Openstack.

Any Member data models without this new feature would have the fields default to the value of null to signify that Octavia's LBaaS service should use the member's protocol port to assess health status.

REST API impact

There are two APIs that will need to be modified, only slightly, to facilitate this change.

Table 1: Octavia LBaaS APIs

Method	URI
POST	/v2.0/lbaas/pools/{pool_id}/members
PUT	/v2.0/lbaas/pools/{pool_id}/members/{member_id}
GET	/v2.0/lbaas/pools/{pool_id}/members/{member_id}

The POST and PUT calls will need two additional fields added to their JSON body data for the request and the JSON response data.

The GET call will need two additional fields as well, however they would only be added to the JSON response data.

The fields to be added to each is:

Table 2: Added Fields

Attribute Name	Type	Access	Default Value	Validation Conversion	Description
monitor_port	int	RW, all	null	int	health check port (optional)
monitor_address	string	RW, all	null	types.IPAddressType()	health check IP address (optional)

Security impact

None

Notifications impact

None

Other end user impact

None

Performance Impact

None

Other deployer impact

None

Developer impact

Other plugins do not have to implement this feature as it is optional due to the default behavior. If they decide to implement this feature, they would just need to supply the protocol port in their POSTs and PUTs to the health monitor APIs.

Implementation

Assignee(s)

Primary assignee: a.amerine

Other contributors: None

Work Items

- Alter the Member Data Model
- Alter Pool Member APIs
- Update API reference documentation to reflect changes
- Write or Alter Unit, Functional, and Tempest Tests to verify new functionality

Dependencies

None

Testing

Integration tests can be written to verify functionality. Generally, it should only require an existing Openstack deployment that is running LBaaS to verify health checks.

Documentation Impact

The REST API impact will need to be addressed in documentation so developers moving forward know about the feature and can use it.

References

- Octavia Roadmap Considerations: Health monitoring on alternate IPs and/or ports (https://wiki.openstack.org/wiki/Octavia/Roadmap)
- RFE Port based HealthMonitor in neutron_lbaas (https://launchpad.net/bugs/1541579)

Align octavia API With Neutron LBaaS API

Problem description

For the octavia API to truly be standalone, it needs to have capability parity with Neutron LBaaS's API. Neutron LBaaS has the luxury of piggy-backing off of Neutron's API. This gives Neutron LBaaS's API resources many capabilities for free. This document is meant to enumerate those capabilities that the octavia API does not possess at the time of this writing.

Proposed change

Complete the tasks enumerated in the *Work Items* section

Alternatives

- Do nothing and keep the status quo

Data model impact

There will be some minor data model changes to octavia in support of this change.

REST API impact

This change will have significant impact to the octavia API.

Security impact

This change will improve octavia security by adding keystone authentication.

Notifications impact

No expected change.

Other end user impact

Users will be able to use the new octavia API endpoint for LBaaS.

Performance Impact

This change may slightly improve performance by reducing the number of software layers requests will traverse before responding to the request.

Other deployer impact

Over time the neutron-lbaas package will be deprecated and deployers will only require octavia for LBaaS.

Developer impact

This will simplify LBaaS development by reducing the number of databases as well as repositories that require updating for LBaaS enhancements.

Implementation

Assignee(s)

blogan diltram johnsom rm_you dougwig

Work Items

Implement the following API Capabilities:

- Keystone Authentication
- Policy Engine
- Pagination
- Quotas
- Filtering lists by query parameter
- Fields by query parameter
- Add the same root API endpoints as n-lbaas
- Support "provider" option in the API to select a driver to spin up a load balancer.
- API Handler layer to become the same as n-lbaas driver layer and allow multiple handlers/drivers.
- Neutron LBaaS V2 driver to octavia API Handler shim layer

Implement the following additional features that n-lbaas maintains:

- OSC extension via a new repository 'python-octaviaclient'

Other Features to be Considered:

- Notifications for resource creating, updating, and deleting.
- Flavors
- Agent namespace driver or some lightweight functional driver.
- Testing octavia with all of the above
- REST API Microversioning

Dependencies

None

Testing

Api tests from neutron-lbaas will be used to validate the new octavia API.

Documentation Impact

The octavia api reference will need to be updated.

References

Vip QoS Policy Application

Problem description

For real cases, the bandwidth of vip should be limited, because the upstream network resource is provided by the ISP or other organizations. That means it is not free. The openstack provider or users should pay for the limited bandwidth, for example, users buy the 50M bandwidth from ISP for openstack environment to access Internet, also it will be used for the connection outside of openstack to access the servers in openstack. And the servers are behind LoadBalance VIP. We cannot offer the whole bandwidth to the servers, as maybe there also are the VMs want to access the external network. So we should take a bandwidth limitation towards vip port.

Also, if the upstream network resource had been used up mostly, we still want the backend servers behind loadbalancer are accessible and stable. The min bandwidth limitation is needed for this scenario.

For more QoS functions, in reality, we can't limit our users or deployers to use loadbalance default drivers, such as haproxy driver and Octavia driver. They may be more concerned about the fields/functions related to QoS, like DSCP markings. They could integrate the third-party drivers which are concerned about these fields.

Proposed change

This spec introduces the Neutron QoS function to meet the requirements. Currently, there are 3 ports(at least) in the loadbalancer created by Octavia. One is from the lb-mgmt-net, the others are from the vip-subnet, called "loadbalancer-LOADBALANCER_ID" and "octavia-lb-vrrp-LOADBALNCER_ID". The first one is vip port, the second one is for vrrp HA, and it will set "allowed_address_pairs" toward vip fixed_ip. The QoS policy should focus on the attached port "octavia-lb-vrrp-LOADBALNCER_ID".

We could apply the Neutron QoS policy to the "octavia-lb-vrrp-LOADBALNCER_ID" ports, whether the topology is active-active or standalone.

There are the following changes:

- Extend a new column named "qos_policy_id" in vip table.
- Extend Octavia API, we need pass the vip-qos-policy-id which created in Neutron into LoadBalancer creation/update.
- Apply QoS policy on vip port in Loadbalancer working flow.

Alternatives

We accept the QoS parameters and implement the QoS function on our own.

Data model impact

In this spec, the QoS function will be provided by Neutron, so Octavia should know the relationship of QoS policies and the vip port of Loadbalancers. There will be some minor data model changes to Octavia in support of this change.

- vip table - *qos_policy_id*: associate QoS policy id with vip port.

REST API impact

Proposed attribute:

```
EXTEND_FIELDS = {
    'vip_qos_policy_id':{'allow_post': True, 'allow_put': True,
                        'validate': {'type:uuid': None},
                        'is_visible': True,
                        'default': None}
}
```

The definition in Octavia is like:: vip_qos_policy_id = wtypes.wsattr(wtypes.UuidType())

Some samples in Loadbalancer creation/update. Users allow pass "vip_qos_policy_id".

Create/Update Loadbalancer Request:

```
POST/PUT /v2.0/lbaas/loadbalancers
{
    "loadbalancer": {
        "name": "loadbalancer1",
        "description": "simple lb",
        "project_id": "b7c1a69e88bf4b21a8148f787aef2081",
        "tenant_id": "b7c1a69e88bf4b21a8148f787aef2081",
        "vip_subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "vip_address": "10.0.0.4",
        "admin_state_up": true,
        "flavor": "a7ae5d5a-d855-4f9a-b187-af66b53f4d04",
        "vip_qos_policy_id": "b61f8b45-e888-4056-94f0-e3d5af96211f"
    }
}

Response:
{
    "loadbalancer": {
        "admin_state_up": true,
        "description": "simple lb",
        "id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c",
        "listeners": [],
        "name": "loadbalancer1",
        "operating_status": "ONLINE",
        "provisioning_status": "ACTIVE",
        "project_id": "b7c1a69e88bf4b21a8148f787aef2081",
```

(continues on next page)

(continued from previous page)

```
        "tenant_id": "b7c1a69e88bf4b21a8148f787aef2081",
        "vip_address": "10.0.0.4",
        "vip_subnet_id": "013d3059-87a4-45a5-91e9-d721068ae0b2",
        "flavor": "a7ae5d5a-d855-4f9a-b187-af66b53f4d04",
        "provider": "sample_provider",
        "pools": [],
        "vip_qos_policy_id": "b61f8b45-e888-4056-94f0-e3d5af96211f"
    }
}
```

Security impact

None

Notifications impact

No expected change.

Other end user impact

Users will be able to specify qos_policy to create/update Loadbalancers.

Performance Impact

- It will be a very short time to cost in loadbalancer creation, as we need validate the input QoS policy.
- The QoS policy in Neutron side will affect the network performance based on the different types of QoS rules.

Other deployer impact

None

Developer impact

TBD.

Implementation

Assignee(s)

zhaobo reedip

Work Items

- Add the DB model and extend the table column.
- Extending Octavia V2 API to accept QoS policy.
- Add QoS application logic into Loadbalancer workflow.
- Add API validation code to validate access/existence of the qos_policy which created in Neutron.
- Add UTs to Octavia.
- Add API tests.
- Update CLI to accept QoS fields.
- Documentation work.

Dependencies

None

Testing

Unit tests, Functional tests, API tests and Scenario tests are necessary.

Documentation Impact

The Octavia API reference will need to be updated.

References

4.5.5 Version 1.1 (queens)

Distributor for L3 Active-Active, N+1 Amphora Setup

> **Attention:** Please review the active-active topology blueprint first (*Active-Active, N+1 Amphorae Setup*)

https://blueprints.launchpad.net/octavia/+spec/l3-active-active

Problem description

This blueprint describes a *L3 active-active* distributor implementation to support the Octavia *active-active-topology*. The *L3 active-active* distributor will leverage the capabilities of a layer 3 Clos network fabric in order to distribute traffic to an *Amphora Cluster* of 1 or more amphoras. Specifically, the *L3 active-active* distributor design will leverage Equal Cost Multipath Load Sharing (ECMP) with anycast routing to achieve traffic distribution across the *Amphora Cluster*. In this reference implementation, the BGP routing protocol will be used to inject anycast routes into the L3 fabric.

In order to scale a single VIP address across multiple active amphoras it is required to have a *distributor* to balance the traffic. By leveraging the existing capabilities of a modern L3 network, we can use the network itself as the *distributor*. This approach has several advantages, which include:

- Traffic will be routed via the best path to the destination amphora. There is no need to add an additional hop (*distributor*) between the network and the amphora.
- The *distributor* is not in the data path and simply becomes a function of the L3 network.
- The performance and scale of the *distributor* is the same as the L3 network.
- Native support for both IPv4 and IPv6, without customized logic for each address family.

  Note: Items marked with [*P2*] refer to lower priority features to be designed / implemented only after initial release.

Proposed change

- Octavia shall implement the *L3 active-active* distributor through a pluggable driver.
- The distributor control plane function (*bgp speaker*) will run inside the amphora and leverage the existing amphora lifecycle manager.
- Each amphora will run a *bgp speaker* in the default namespace in order to announce the anycast VIP into the L3 fabric. BGP peering and announcements will occur over the lb-mgmt-net network. The anycast VIP will get advertised as a /32 or /128 route with a next-hop of the front-end IP assigned to the amphora instance. The front-end network IPs must be directly routable from the L3 fabric, such as in the provider networking model.
- Octavia shall implement the ability to specify an anycast VIP/subnet and front-end subnet (provider network) when creating a new load balancer. The amphora will have ports on three networks (anycast, front-end, management). The anycast VIP will get configured on the loopback interface inside the *amphora-haproxy* network namespace.
- The operator shall be able to define a *bgp peer profile*, which includes the required metadata for the amphora to establish a bgp peering session with the L3 fabric. The bgp peering information will be passed into the amphora-agent configuration file via config drive during boot. The amphora will use the bgp peering information to establish a BGP peer and announce its anycast VIP.
- [*P2*] Add the option to allow the *bgp speaker* to run on a dedicated amphora instance that is not running the software load balancer (HAProxy). In this model a dedicated *bgp speaker* could advertise anycast VIPs for one or more amphoras. Each BGP speaker (peer) can only announce a single next-hop route for an anycast VIP. In order to perform ECMP load sharing, multiple dedicated amphoras running bgp speakers will be required, each of them would then announce a different next-hop address for the anycast VIP. Each next-hop address is the front-end (provider network) IP of an amphora instance running the software load balancer.

- [*P2*] The *Amphora Cluster* will provide resilient flow handling in order to handle ECMP group flow remapping events and support amphora connection draining.
- [*P2*] Support Floating IPs (FIPs). In order to support FIPs the existing Neutron *floatingips* API would need to be extended. This will be described in more detail in a separate spec in the Neutron project.

Architecture

High-level Topology Description

The below diagram shows the interaction between 2 .. n amphora instances from each tenant and how they interact with the L3 network distributor.

```
                 Management                                        Front-End
Internet         Network                                           Networks
(World)                                                           (provider)

                                       Amphora of Tenant A

                     MGMTns: defaultns: amphora-haproxyf.e.
                            IP ------------------------------ IP
                               BGP        Anycast VIP
                                   Speaker     (loopback)

                                       |
                                       |
                Peering Session   1..*  |
             ---------------------------+
                {anycast VIP}/32 next-hop {f.e. IP}


                                      Amphora of Tenant B

             MGMTns: defaultns: amphora-haproxyf.e.
                            IP ------------------------------ IP
                               BGP        Anycast VIP
                                   Speaker     (loopback)

 Distributor                           |
 (L3 Network)                          |
                Peering Session   1..*  |
             ---------------------------+
                {anycast VIP}/32 next-hop {f.e. IP}


                                      Amphora of Tenant C

                     MGMTns: defaultns: amphora-haproxyf.e.
                             IP ------------------------------ IP
```

(continues on next page)

(continued from previous page)

```
                   BGP        Anycast VIP
                       Speaker     (loopback)

                           |
                           |
   Peering Session   1..*  |
---------------------------+
   {anycast VIP}/32 next-hop {f.e. IP}

Anycast
        1..*                      Network
```

- Whenever a new active-active amphora is instantiated it will create BGP peering session(s) over the lb-mgmt-net to the L3 fabric. The BGP peer will need to have a neighbor definition in order to allow the peering sessions from the amphoras. In order to ease configuration, a neighbor statement allowing peers from the entire lb-mgmt-net IP prefix range can be defined: `neighbor 10.10.10.0/24`
- The BGP peer IP can either be a route reflector (RR) or any other network device that will redistribute routes learned from the amphora BGP speaker. In order to help scaling, it is possible to peer with the ToR switch based on the rack the amphora instance is provisioned in. The configuration can be simplified by creating an `anycast loopback interface` on each ToR switch, which will provide a consistent BGP peer IP regardless of which rack or hypervisor is hosting the amphora instance.
- Once a peering session is established between an amphora and the L3 fabric, the amphora will need to announce its anycast VIP with a next-hop address of its front-end network IP. The front-end network IP (provider) must be routable and reachable from the L3 network in order to be used.
- In order to leverage ECMP for distributing traffic across multiple amphoras, multiple equal-cost routes must be installed into the network for the anycast VIP. This requires the L3 network to have `Multipath BGP` enabled, so BGP installs multiple paths and does not select a single best path.
- After the amphoras in a cluster are initialized there will be an ECMP group with multiple equal-cost routes for the anycast VIP. The data flow for traffic is highlighted below:
  1. Traffic will ingress into the L3 network fabric with a destination IP address of the anycast VIP.
  2. If this is a new flow, the flow will get hashed to one of the next-hop addresses in the ECMP group.
  3. The packet will get sent to the front-end IP address of the amphora instance that was selected from the above step.
  4. The amphora will accept the packet and send it to the back-end server over the front-end network or a directly attached back-end (tenant) network attached to the amphora.
  5. The amphora will receive the response from the back-end server and forward it on to the next-hop gateway of front-end (provider) network using the anycast VIP as the source IP address.
  6. All subsequent packets belonging to the same flow will get routed through the same path.
- Adding or removing members to a L3 active-active amphora cluster will result in flow remapping,

as different paths will be selected due to rehashing. It is recommended to enable the `resilient hashing` feature on ECMP groups in order to minimize flow remapping.

Distributor (BGP Speaker) Lifecycle

The below diagram shows the interaction between an amphora instance that is serving as a distributor and the L3 network. In this example we are peering with the ToR switch in order to disseminate anycast VIP routes into the L3 network.

```
+------------------------------------------------+
|       Initialize Distributor on Amphora        |
+------------------------------------------------+
|                                                |
| +---------------+            +---------------+ |
| |1              |            |4              | |
| |    Amphora    |            |   Ready to    | |
| |    (boot)     |            |   announce    | |
| |               |            |    VIP(s)     | |
| +-------+-------+            +-------+-------+ |
|         |                            ^         |
|         |                            |         |
|         |                            |         |
|         |                            |         |
|         |                            |         |
|         v                            |         |
| +-------+-------+            +-------+-------+ |
| |2              |            |3  Establish   | |
| |  Read Config  |            | BGP connection| |
| |     Drive     +----------->+    to ToR(s)   | |
| |  (BGP Config) |            | (BGP Speaker) | |
| +---------------+            +---------------+ |
|                                                |
+------------------------------------------------+

+------------------------------------------------+
| Register AMP to Distributor or Listener Start   |
+------------------------------------------------+
|                                                |
| +---------------+            +---------------+ |
| |5              |            |8              | |
| |    Amphora    |            |    Amphora    | |
| |  BGP Speaker  |            | (Receives VIP | |
| |(Announce VIP) |            |   Traffic)    | |
| +-------+-------+            +-------+-------+ |
|         |                            ^         |
|         |                            |         |
|         |BGP Peering                 |         |
|         |Session(s)                  |         |
|         |                            |         |
|         v                            |         |
```

(continues on next page)

(continued from previous page)

```
| +-------+-------+            +-------+-------+ |
| |6              |            |7              | |
| |    ToR(s)     |            |   L3 Fabric   | |
| |(Injects Route +----------->+ Accepts Route | |
| | into Fabric)  |            |    (ECMP)     | |
| +---------------+            +---------------+ |
|                                                |
+------------------------------------------------+


+------------------------------------------------+
| Unregister AMP to Distributor or Listener Stop |
+------------------------------------------------+
|                                                |
| +---------------+            +---------------+ |
| |9              |            |12             | |
| |    Amphora    |            |    Amphora    | |
| |  BGP Speaker  |            |(No longer sent| |
| |(Withdraw VIP) |            | VIP traffic)  | |
| +-------+-------+            +-------+-------+ |
|         |                            ^         |
|         |                            |         |
|         |BGP Peering                 |         |
|         |Session(s)                  |         |
|         |                            |         |
|         v                            |         |
| +-------+-------+            +-------+-------+ |
| |10             |            |11             | |
| |    ToR(s)     |            |   L3 Fabric   | |
| |(Removes Route +----------->+ Removes Route | |
| | from Fabric)  |            |    (ECMP)     | |
| +---------------+            +---------------+ |
|                                                |
+------------------------------------------------+
```

1. The amphora gets created and is booted. In this example, the amphora will perform both the load balancing (HAProxy) and L3 Distributor function (BGP Speaker).
2. The amphora will read in the BGP configuration information from the config drive and configure the BGP Speaker to peer with the ToR switch.
3. The BGP Speaker process will start and establish a BGP peering session with the ToR switch.
4. Once the BGP peering session is active, the amphora is ready to advertise its anycast VIP into the network with a next-hop of its front-end IP address.
5. The BGP speaker will communicate using the BGP protocol and send a BGP "announce" message to the ToR switch in order to announce a VIP route. If the amphora is serving as both a load balancer and distributor the announcement will happen on listener start. Otherwise the announce will happen on a register amphora request to the distributor.
6. The ToR switch will learn this new route and advertise it into the L3 fabric. At this point the L3 fabric will know of the new VIP route and how to reach it (via the ToR that just announced it).

7. The L3 fabric will create an ECMP group if it has received multiple route advertisements for the same anycast VIP. This will result in a single VIP address with multiple next-hop addresses.
8. Once the route is accepted by the L3 fabric, traffic will get distributed to the recently registered amphora (HAProxy).
9. The BGP speaker will communicate using the BGP protocol and send a BGP "withdraw" message to the ToR switch in order to withdraw a VIP route. If the amphora is serving as both a load balancer and distributor the withdrawal will happen on listener stop. Otherwise the withdraw will happen on an unregister amphora request to the distributor.
10. The ToR switch will tell the L3 fabric over BGP that the anycast VIP route for the amphora being unregistered is no longer valid.
11. The L3 fabric will remove the VIP address with the next-hop address to the amphora (HAProxy) being unregistered. It will keep all other existing VIP routes to other amphora (HAProxy) instances until they are explicitly unregistered.
12. Once the route is removed the amphora (HAProxy) will no longer receive any traffic for the VIP.

Alternatives

TBD

Data model impact

Add the following columns to the existing `vip` table:

- **distributor_id `(String(36) , nullable=True)`** ID of the distributor responsible for distributing traffic for the corresponding VIP.

Add table `distributor` with the following columns:

- **id `(String(36) , nullable=False)`** ID of Distributor instance.
- **distributor_type `(String(36) , nullable=False)`** Type of distributor `L3_BGP`.
- **status `(String(36) , nullable=True)`** Provisioning status.

Update existing table `amphora`. An amphora can now serve as a distributor, lb, or both. The vrrp_* tables will be renamed to frontend_* in order to make the purpose of this interface more apparent and to better represent other use cases besides active/standy.

- **load_balancer_id `(String(36) , nullable=True)`** This will be set to null if this amphora is a dedicated distributor and should not run HAProxy.
- **service_type `(String(36) , nullable=True)`** New field added to the amphora table in order to describe the type of amphora. This field is used to describe the function (service) the amphora provides. For example, if this is a dedicated distributor the service type would be set to "distributor".
- **frontend_ip `(String(64) , nullable=True)`** New name for former vrrp_ip field. This is the primary IP address inside the amphora-haproxy namespace used for L3 communication to back-end members.
- **frontend_subnet_id `(String(36) , nullable=True)`** New field added to the amphora table, which is the neutron subnet id of the front-end network connected to the amphora.

- **frontend_port_id** (`String(36)` , `nullable=True`) New name for former vrrp_port_id field. This represents the neutron port ID of a port attached to the front-end network. It should no longer be assumed that the front-end subnet is the same as the VIP subnet.
- **frontend_interface** (`String(16)` , `nullable=True`) New name for former vrrp_interface field.
- **frontend_id** (`Integer` , `nullable=True`) New name for former vrrp_id field.
- **frontend_priority** (`Integer` , `nullable=True`) New name for former vrrp_priority field.

Use existing table `amphora_health` with the following columns:

- **amphora_id** (`String(36)` , `nullable=False`) ID of amphora instance running lb and/or implementing distributor function.
- **last_update** (`DateTime` , `nullable=False`) Last time amphora heartbeat was received by a health monitor.
- **busy** (`Boolean` , `nullable=False`) Field indicating a create / delete or other action is being conducted on the amphora instance (ie. to prevent a race condition when multiple health managers are in use).

Add table `amphora_registration` with the below columns. This table determines the role of the amphora. The amphora can be dedicated as a distributor, load balancer, or perform a combined role of load balancing and distributor. A distributor amphora can be registered to multiple load balancers.

- **amphora_id** (`String(36)` , `nullable=False`) ID of Amphora instance.
- **load_balancer_id** (`String(36)` , `nullable=False`) ID of load balancer.
- **distributor_id** (`String(36)` , `nullable=True`) ID of Distributor instance.

Add table `distributor_l3_bgp_speaker` with the following columns:

- **id** (`String(36)` , `nullable=False`) ID of the BGP Speaker.
- **ip_version** (`Integer` , `nullable=False`) Protocol version of the BGP speaker. IP version `4` or `6`.
- **local_as** (`Integer` , `nullable=False`) Local AS number for the BGP speaker.

Add table `distributor_l3_bgp_peer` with the following columns:

- **id** (`String(36)` , `nullable=False`) ID of the BGP peer.
- **peer_ip** (`String(64)` , `nullable=False`) The IP address of the BGP neighbor.
- **remote_as** (`Integer` , `nullable=False`) Remote AS of the BGP peer.
- **auth_type** (`String(16)` , `nullable=True`) Authentication type, such as `md5`. An additional parameter will need to be set in the octavia configuration file by the admin to set the md5 authentication password that will be used with the md5 auth type.
- **ttl_hops** (`Integer` , `nullable=True`) Number of hops between speaker and peer for ttl security `1-254`.
- **hold_time** (`Integer` , `nullable=True`) Amount of time in seconds that can elapse between messages from peer.
- **keepalive_interval** (`Integer` , `nullable=True`) How often to send keep alive packets in seconds.

Add table `distributor_l3_bgp_peer_registration` with the following columns:

- **distributor_l3_bgp_speaker_id (`String(36) , nullable=False`)** ID of the BGP Speaker.
- **distributor_l3_bgp_peer_id (`String(36) , nullable=False`)** ID of the BGP peer.

Add table `distributor_l3_amphora_bgp_speaker_registration` with the following columns:

- **distributor_l3_bgp_speaker_id (`String(36) , nullable=False`)** ID of the BGP Speaker.
- **amphora_id (`String(36) , nullable=False`)** ID of amphora instance that the BGP speaker will run on.

Add table `distributor_l3_amphora_vip_registration` with the following columns:

- **amphora_id (`String(36) , nullable=False`)** ID of the distributor amphora instance.
- **load_balancer_id (`String(36) , nullable=False`)** The ID of the load balancer. This will be used to get the VIP IP address.
- **nexthop_ip (`String(64) , nullable=False`)** The amphora front-end network IP used to handle VIP traffic. This is the next-hop address that will be advertised for the VIP. This does not have to be an IP address of an amphora, as it could be external such as for UDP load balancing.
- **distributor_l3_bgp_peer_id (`String(36) , nullable=True`)** The BGP peer we will announce the anycast VIP to. If not specified, we will announce over all peers.

REST API impact

- Octavia API -- Allow the user to specify a separate VIP/subnet and front-end subnet (provider network) when creating a new load balancer. Currently the user can only specify the VIP subnet, which results in both the VIP and front-end network being on the same subnet.
- Extended Amphora API -- The L3 BGP distributor driver will call the extended amphora API in order to implement the control plane (BGP) and advertise new anycast VIP routes into the network.

The below extended amphora API calls will be implemented for amphoras running as a dedicated distributor:

1. `Register Amphora`

   This call will result in the BGP speaker announcing the anycast VIP into the L3 network with a next-hop of the front-end IP of the amphora being registered. Prior to this call, the load balancing amphora will have to configure the anycast VIP on the loopback interface inside the amphora-haproxy namespace.

   - **amphora_id** ID of the amphora running the load balancer to register.
   - **vip_ip** The VIP IP address.
   - **nexthop_ip** The amphora's front-end network IP address used to handle anycast VIP traffic.
   - **peer_id** ID of the peer that will be used to announce the anycast VIP. If not specified, VIP will be announced across all peers.

2. `Unregister Amphora`

The BGP speaker will withdraw the anycast VIP route for the specified amphora from the L3 network. After the route is withdrawn, the anycast VIP IP will be removed from the loopback interface on the load balancing amphora.

- **amphora_id** ID of the amphora running the load balancer to unregister.
- **vip_ip** The VIP IP address.
- **nexthop_ip** The amphora's front-end network IP Address used to handle anycast VIP traffic.
- **peer_id** ID of the peer that will be used to withdraw the anycast VIP. If not specified, route will be withdrawn from all peers.

3. `List Amphora`

   Will return a list of all amphora IDs and their anycast VIP routes currently being advertised by the BGP speaker.

4. [*P2*] `Drain Amphora`

   All new flows will get redirected to other members of the cluster and existing flows will be drained. Once the active flows have been drained, the BGP speaker will withdraw the anycast VIP route from the L3 network and unconfigure the VIP from the lo interface.

5. [*P2*] `Register VIP`

   This call will be used for registering anycast routes for non-amphora endpoints, such as for UDP load balancing.

   - **vip_ip** The VIP IP address.
   - **nexthop_ip** The nexthop network IP Address used to handle anycast VIP traffic.
   - **peer_id** ID of the peer that will be used to announce the anycast VIP. If not specified, route will be announced from all peers.

6. [*P2*] `Unregister VIP`

   This call will be used for unregistering anycast routes for non-amphora endpoints, such as for UDP load balancing.

   - **vip_ip** The VIP IP address.
   - **nexthop_ip** The nexthop network IP Address used to handle anycast VIP traffic.
   - **peer_id** ID of the peer that will be used to withdraw the anycast VIP. If not specified, route will be withdrawn from all peers.

6. [*P2*] `List VIP`

   Will return a list of all non-amphora anycast VIP routes currently being advertised by the BGP speaker.

Security impact

The distributor inherently supports multi-tenancy, as it is simply providing traffic distribution across multiple amphoras. Network isolation on a per tenant basis is handled by the amphoras themselves, as they service only a single tenant. Further isolation can be provided by defining separate anycast network(s) on a per tenant basis. Firewall or ACL policies can then be built around these prefixes.

To further enhance BGP security, route-maps, prefix-lists, and communities to control what routes are allowed to be advertised in the L3 network from a particular BGP peer can be used. MD5 password and GTSM can provide additional security to limit unauthorized BGP peers to the L3 network.

Notifications impact

Other end user impact

Performance Impact

Other deployer impact

Developer impact

Implementation

Assignee(s)

Work Items

Dependencies

Testing

- Unit tests with tox.
- Function tests with tox.

Documentation Impact

The API-Ref documentation will need to be updated for load balancer create. An additional optional parameter frontend_network_id will be added. If set, this parameter will result in the primary interface inside the amphora-haproxy namespace getting created on the specified network. Default behavior is to provision this interface on the VIP subnet.

References

- Active-Active Topology

Enable Provider Driver Support

Specification Table of Contents

- *Enable Provider Driver Support*
    - *Problem description*
    - *Proposed change*
        - *Driver Entry Points*
        - *Octavia Provider Driver API*
            - *Load balancer*
            - *Listener*
            - *Pool*
            - *Member*
            - *Health Monitor*
            - *L7 Policy*
            - *L7 Rule*
            - *Flavor*
            - *Exception Model*
        - *Driver Support Library*
            - *Update provisioning and operating status API*
            - *Update statistics API*
            - *Get Resource Support*
            - *API Exception Model*
        - *Alternatives*
        - *Data model impact*
        - *REST API impact*
        - *Security impact*
        - *Notifications impact*
        - *Other end user impact*
        - *Performance Impact*
        - *Other deployer impact*

    - *Developer impact*
  - *Implementation*
    - *Assignee(s)*
    - *Work Items*
  - *Dependencies*
  - *Testing*
  - *Documentation Impact*
  - *References*

https://storyboard.openstack.org/#!/story/1655768

Provider drivers are implementations that give Octavia operators a choice of which load balancing systems to use in their Octavia deployment. Currently, the default Octavia driver is the only one available. Operators may want to employ other load balancing implementations, including hardware appliances, in addition to the default Octavia driver.

Problem description

Neutron LBaaS v2 supports a *provider* parameter, giving LBaaS users a way to direct LBaaS requests to a specific backend driver. The Octavia API includes a *provider* parameter as well, but currently supports one provider, the Octavia driver. Adding support for other drivers is needed. With this in place, operators can configure load balancers using multiple providers, either the Octavia default or others.

Proposed change

Available drivers will be enabled by entries in the Octavia configuration file. Drivers will be loaded via stevedore and Octavia will communicate with drivers through a standard class interface defined below. Most driver functions will be asynchronous to Octavia, and Octavia will provide a library of functions that give drivers a way to update status and statistics. Functions that are synchronous are noted below.

Octavia API functions not listed here will continue to be handled by the Octavia API and will not call into the driver. Examples would be show, list, and quota requests.

Driver Entry Points

Provider drivers will be loaded via stevedore. Drivers will have an entry point defined in their setup tools configuration using the Octavia driver namespace "octavia.api.drivers". This entry point name will be used to enable the driver in the Octavia configuration file and as the "provider" parameter users specify when creating a load balancer. An example for the octavia reference driver would be:

```
octavia = octavia.api.drivers.octavia.driver:OctaviaDriver
```

Octavia Provider Driver API

Provider drivers will be expected to support the full interface described by the Octavia API, currently v2.0. If a driver does not implement an API function, drivers should fail a request by raising a `NotImplementedError` exception. If a driver implements a function but does not support a particular option passed in by the caller, the driver should raise an `UnsupportedOptionError`.

It is recommended that drivers use the jsonschema package or voluptuous to validate the request against the current driver capabilities.

See the *Exception Model* below for more details.

Note: Driver developers should refer to the official *Octavia API reference <https://docs.openstack.org/api-ref/load-balancer/v2/index.html>* document for details of the fields and expected outcome of these calls.

Load balancer

- **create**

  Creates a load balancer.

  Octavia will pass in the load balancer object with all requested settings.

  The load balancer will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the load balancer to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The load balancer python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The provider will be removed as this is used for driver selection.
  2. The flavor will be expanded from the provided ID to be the full dictionary representing the flavor metadata.

  Load balancer object

  As of the writing of this specification the create load balancer object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the resource.
flavor	dict	The flavor keys and values.
listeners	list	A list of *Listener objects*.
loadbalancer_id	string	ID of load balancer to create.
name	string	Human-readable name of the resource.
pools	list	A list of *Pool object*.
project_id	string	ID of the project owning this resource.
vip_address	string	The IP address of the Virtual IP (VIP).
vip_network_id	string	The ID of the network for the VIP.
vip_port_id	string	The ID of the VIP port.
vip_qos_policy_id	string	The ID of the qos policy for the VIP.
vip_subnet_id	string	The ID of the subnet for the VIP.

The driver is expected to validate that the driver supports the request and raise an exception if the request cannot be accepted.

VIP port creation

Some provider drivers will want to create the Neutron port for the VIP, and others will want Octavia to create the port instead. In order to support both use cases, the create_vip_port() method will ask provider drivers to create a VIP port. If the driver expects Octavia to create the port, the driver will raise a NotImplementedError exception. Octavia will call this function before calling loadbalancer_create() in order to determine if it should create the VIP port. Octavia will call create_vip_port() with a loadbalancer ID and a partially defined VIP dictionary. Provider drivers that support port creation will create the port and return a fully populated VIP dictionary.

VIP dictionary

Name	Type	Description
project_id	string	ID of the project owning this resource.
vip_address	string	The IP address of the Virtual IP (VIP).
vip_network_id	string	The ID of the network for the VIP.
vip_port_id	string	The ID of the VIP port.
vip_qos_policy_id	string	The ID of the qos policy for the VIP.
vip_subnet_id	string	The ID of the subnet for the VIP.

Creating a Fully Populated Load Balancer

If the "listener" option is specified, the provider driver will iterate through the list and create all of the child objects in addition to creating the load balancer instance.

- **delete**

  Removes an existing load balancer.

  Octavia will pass in the load balancer object and cascade boolean as parameters.

  The load balancer will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

The API includes an option for cascade delete. When cascade is set to True, the provider driver will delete all child objects of the load balancer.

- **failover**

  Performs a failover of a load balancer.

  Octavia will pass in the load balancer ID as a parameter.

  The load balancer will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the load balancer to either `ACTIVE` if successfully failed over, or `ERROR` if not failed over.

  Failover can mean different things in the context of a provider driver. For example, the Octavia driver replaces the current amphora(s) with another amphora. For another provider driver, failover may mean failing over from an active system to a standby system.

- **update**

  Modifies an existing load balancer using the values supplied in the load balancer object.

  Octavia will pass in the original load balancer object which is the baseline for the update, and a load balancer object with the fields to be updated.

  As of the writing of this specification the update load balancer object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the resource.
loadbalancer_id	string	ID of load balancer to update.
name	string	Human-readable name of the resource.
vip_qos_policy_id	string	The ID of the qos policy for the VIP.

  The load balancer will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the load balancer to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

  The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):

  def create_vip_port(self, loadbalancer_id, vip_dictionary):
      """Creates a port for a load balancer VIP.

      If the driver supports creating VIP ports, the driver will create a
      VIP port and return the vip_dictionary populated with the vip_port_id.
      If the driver does not support port creation, the driver will raise
      a NotImplementedError.

      :param: loadbalancer_id (string): ID of loadbalancer.
      :param: vip_dictionary (dict): The VIP dictionary.
      :returns: VIP dictionary with vip_port_id.
      :raises DriverError: An unexpected error occurred in the driver.
```

(continues on next page)

(continued from previous page)

```
        :raises NotImplementedError: The driver does not support creating
          VIP ports.
        """
        raise NotImplementedError()

    def loadbalancer_create(self, loadbalancer):
        """Creates a new load balancer.

        :param loadbalancer (object): The load balancer object.
        :return: Nothing if the create request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: The driver does not support create.
        :raises UnsupportedOptionError: The driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()

    def loadbalancer_delete(self, loadbalancer, cascade=False):
        """Deletes a load balancer.

        :param loadbalancer (object): The load balancer object.
        :param cascade (bool): If True, deletes all child objects (listeners,
          pools, etc.) in addition to the load balancer.
        :return: Nothing if the delete request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        """
        raise NotImplementedError()

    def loadbalancer_failover(self, loadbalancer_id):
        """Performs a fail over of a load balancer.

        :param loadbalancer_id (string): ID of the load balancer to failover.
        :return: Nothing if the failover request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises: NotImplementedError if driver does not support request.
        """
        raise NotImplementedError()

    def loadbalancer_update(self, old_loadbalancer, new_loadbalancer):
        """Updates a load balancer.

        :param old_loadbalancer (object): The baseline load balancer object.
        :param new_loadbalancer (object): The updated load balancer object.
        :return: Nothing if the update request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: The driver does not support request.
        :raises UnsupportedOptionError: The driver does not
          support one of the configuration options.
```

(continues on next page)

(continued from previous page)

```
        """
        raise NotImplementedError()
```

Listener

- **create**

  Creates a listener for a load balancer.

  Octavia will pass in the listener object with all requested settings.

  The listener will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the listener to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The listener python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.
  2. The default_tls_container_ref will be expanded and provided to the driver in pkcs12 format.
  3. The sni_container_refs will be expanded and provided to the driver in pkcs12 format.

  Listener object

  As of the writing of this specification the create listener object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
connection_limit	int	The max number of connections permitted for this listener. Default is -1, which is infinite connections.
default_pool	object	A *Pool object*.
default_pool_id	string	The ID of the pool used by the listener if no L7 policies match.
default_tls_container_data	dict	A *TLS container* dict.
default_tls_container_refs	string	The reference to the secrets container.
description	string	A human-readable description for the listener.
insert_headers	dict	A dictionary of optional headers to insert into the request before it is sent to the backend member. See *Supported HTTP Header Insertions*. Keys and values are specified as strings.
l7policies	list	A list of *L7policy objects*.
listener_id	string	ID of listener to create.
loadbalancer_id	string	ID of load balancer.
name	string	Human-readable name of the listener.
protocol	string	Protocol type: One of HTTP, HTTPS, TCP, or TERMINATED_HTTPS.
protocol_port	int	Protocol port number.
sni_container_data	list	A list of *TLS container* dict.
sni_container_refs	list	A list of references to the SNI secrets containers.
timeout_client_data	int	Frontend client inactivity timeout in milliseconds.
timeout_member_connect	int	Backend member connection timeout in milliseconds.
timeout_member_data	int	Backend member inactivity timeout in milliseconds.
timeout_tcp_inspect	int	Time, in milliseconds, to wait for additional TCP packets for content inspection.

As of the writing of this specification the TLS container dictionary contains the following:

Key	Type	Description
certificate	string	The PEM encoded certificate.
intermediates	List	A list of intermediate PEM certificates.
primary_cn	string	The primary common name of the certificate.
private_key	string	The PEM encoded private key.

As of the writing of this specification the Supported HTTP Header Insertions are:

Key	Type	Description
X-Forwarded-For	bool	When True a X-Forwarded-For header is inserted into the request to the backend member that specifies the client IP address.
X-Forwarded-Port	int	A X-Forwarded-Port header is inserted into the request to the backend member that specifies the integer provided. Typically this is used to indicate the port the client connected to on the load balancer.

Creating a Fully Populated Listener

If the "default_pool" or "l7policies" option is specified, the provider driver will create all of the child objects in addition to creating the listener instance.

- **delete**

  Deletes an existing listener.

  Octavia will pass the listener object as a parameter.

  The listener will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

- **update**

  Modifies an existing listener using the values supplied in the listener object.

  Octavia will pass in the original listener object which is the baseline for the update, and a listener object with the fields to be updated.

  As of the writing of this specification the update listener object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
connection_limit	int	The max number of connections permitted for this listener. Default is -1, which is infinite connections.
default_pool_id	string	The ID of the pool used by the listener if no L7 policies match.
default_tls_container_data	dict	A *TLS container* dict.
default_tls_container_refs	string	The reference to the secrets container.
description	string	A human-readable description for the listener.
insert_headers	dict	A dictionary of optional headers to insert into the request before it is sent to the backend member. See *Supported HTTP Header Insertions*. Keys and values are specified as strings.
listener_id	string	ID of listener to update.
name	string	Human-readable name of the listener.
sni_container_data	list	A list of *TLS container* dict.
sni_container_refs	list	A list of references to the SNI secrets containers.
timeout_client_data	int	Frontend client inactivity timeout in milliseconds.
timeout_member_connect	int	Backend member connection timeout in milliseconds.
timeout_member_data	int	Backend member inactivity timeout in milliseconds.
timeout_tcp_inspect	int	Time, in milliseconds, to wait for additional TCP packets for content inspection.

The listener will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the listener to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def listener_create(self, listener):
      """Creates a new listener.

      :param listener (object): The listener object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
    raise NotImplementedError()
```

(continues on next page)

(continued from previous page)

```
def listener_delete(self, listener):
    """Deletes a listener.

    :param listener (object): The listener object.
    :return: Nothing if the delete request was accepted.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: if driver does not support request.
    """
    raise NotImplementedError()

def listener_update(self, old_listener, new_listener):
    """Updates a listener.

    :param old_listener (object): The baseline listener object.
    :param new_listener (object): The updated listener object.
    :return: Nothing if the update request was accepted.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: if driver does not support request.
    :raises UnsupportedOptionError: if driver does not
      support one of the configuration options.
    """
    raise NotImplementedError()
```

Pool

- **create**

  Creates a pool for a load balancer.

  Octavia will pass in the pool object with all requested settings.

  The pool will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the pool to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The pool python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.

  Pool object

  As of the writing of this specification the create pool object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the pool.
healthmonitor	object	A *Healthmonitor object*.
lb_algorithm	string	Load balancing algorithm: One of ROUND_ROBIN, LEAST_CONNECTIONS, or SOURCE_IP.
loadbalancer_id	string	ID of load balancer.
listener_id	string	ID of listener.
members	list	A list of *Member objects*.
name	string	Human-readable name of the pool.
pool_id	string	ID of pool to create.
protocol	string	Protocol type: One of HTTP, HTTPS, PROXY, or TCP.
session_persistence	dict	Defines session persistence as one of {'type': <'HTTP_COOKIE' | 'SOURCE_IP'>} OR {'type': 'APP_COOKIE', 'cookie_name': <cookie_name>}

- **delete**

  Removes an existing pool and all of its members.

  Octavia will pass the pool object as a parameter.

  The pool will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

- **update**

  Modifies an existing pool using the values supplied in the pool object.

  Octavia will pass in the original pool object which is the baseline for the update, and a pool object with the fields to be updated.

  As of the writing of this specification the update pool object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the pool.
lb_algorithm	string	Load balancing algorithm: One of ROUND_ROBIN, LEAST_CONNECTIONS, or SOURCE_IP.
name	string	Human-readable name of the pool.
pool_id	string	ID of pool to update.
session_persistence	dict	Defines session persistence as one of {'type': <'HTTP_COOKIE' | 'SOURCE_IP'>} OR {'type': 'APP_COOKIE', 'cookie_name': <cookie_name>}

The pool will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the pool to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def pool_create(self, pool):
      """Creates a new pool.

      :param pool (object): The pool object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def pool_delete(self, pool):
      """Deletes a pool and its members.

      :param pool (object): The pool object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      """
      raise NotImplementedError()

  def pool_update(self, old_pool, new_pool):
      """Updates a pool.

      :param old_pool (object): The baseline pool object.
      :param new_pool (object): The updated pool object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()
```

Member

- **create**

  Creates a member for a pool.

  Octavia will pass in the member object with all requested settings.

  The member will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the member to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The member python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The project_id will be removed, if present, as this field is now deprecated. The member will inherit the project_id from the parent load balancer.

  Member object

  As of the writing of this specification the create member object may contain the following:

Name	Type	Description
address	string	The IP address of the backend member to receive traffic from the load balancer.
admin_state_up	bool	Admin state: True if up, False if down.
backup	bool	Is the member a backup? Backup members only receive traffic when all non-backup members are down.
member_id	string	ID of member to create.
monitor_address	string	An alternate IP address used for health monitoring a backend member.
monitor_port	int	An alternate protocol port used for health monitoring a backend member.
name	string	Human-readable name of the member.
pool_id	string	ID of pool.
protocol_port	int	The port on which the backend member listens for traffic.
subnet_id	string	Subnet ID.
weight	int	The weight of a member determines the portion of requests or connections it services compared to the other members of the pool. For example, a member with a weight of 10 receives five times as many requests as a member with a weight of 2. A value of 0 means the member does not receive new connections but continues to service existing connections. A valid value is from 0 to 256. Default is 1.

- **delete**

  Removes a pool member.

Octavia will pass the member object as a parameter.

The member will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

- **update**

  Modifies an existing member using the values supplied in the listener object.

  Octavia will pass in the original member object which is the baseline for the update, and a member object with the fields to be updated.

  As of the writing of this specification the update member object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
backup	bool	Is the member a backup? Backup members only receive traffic when all non-backup members are down.
member_id	string	ID of member to update.
monitor_address	string	An alternate IP address used for health monitoring a backend member.
monitor_port	int	An alternate protocol port used for health monitoring a backend member.
name	string	Human-readable name of the member.
weight	int	The weight of a member determines the portion of requests or connections it services compared to the other members of the pool. For example, a member with a weight of 10 receives five times as many requests as a member with a weight of 2. A value of 0 means the member does not receive new connections but continues to service existing connections. A valid value is from 0 to 256. Default is 1.

  The member will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the member to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

  The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

- **batch update**

  Set the state of members for a pool in one API call. This may include creating new members, deleting old members, and updating existing members. Existing members are matched based on address/port combination.

  For example, assume a pool currently has two members. These members have the following address/port combinations: '192.0.2.15:80' and '192.0.2.16:80'. Now assume a PUT request is made that includes members with address/port combinations: '192.0.2.16:80' and '192.0.2.17:80'. The member '192.0.2.15:80' will be deleted because it was not in the request. The member '192.0.2.16:80' will be updated to match the request data for that member, because it was matched. The member '192.0.2.17:80' will be created, because no such member existed.

The members will be in the `PENDING_CREATE`, `PENDING_UPDATE`, or `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the members to either `ACTIVE` or `DELETED` if successfully updated, or `ERROR` if the update was not successful.

The batch update method will supply a list of *Member objects*. Existing members not in this list should be deleted, existing members in the list should be updated, and members in the list that do not already exist should be created.

Abstract class definition

```
class Driver(object):
  def member_create(self, member):
      """Creates a new member for a pool.

      :param member (object): The member object.

      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
  raise NotImplementedError()

  def member_delete(self, member):

      """Deletes a pool member.

      :param member (object): The member object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      """
      raise NotImplementedError()

  def member_update(self, old_member, new_member):

      """Updates a pool member.

      :param old_member (object): The baseline member object.
      :param new_member (object): The updated member object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def member_batch_update(self, members):
      """Creates, updates, or deletes a set of pool members.
```

(continues on next page)

(continued from previous page)

```
    :param members (list): List of member objects.
    :return: Nothing if the create request was accepted.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: if driver does not support request.
    :raises UnsupportedOptionError: if driver does not
      support one of the configuration options.
    """
    raise NotImplementedError()
```

Health Monitor

- **create**

  Creates a health monitor on a pool.

  Octavia will pass in the health monitor object with all requested settings.

  The health monitor will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the health monitor to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The healthmonitor python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.

  Healthmonitor object

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
delay	int	The interval, in seconds, between health checks.
expected_codes	string	The expected HTTP status codes to get from a successful health check. This may be a single value, a list, or a range.
healthmonitor_id	string	ID of health monitor to create.
http_method	string	The HTTP method that the health monitor uses for requests. One of CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT, or TRACE.
max_retries	int	The number of successful checks before changing the operating status of the member to ONLINE.
max_retries_down	int	The number of allowed check failures before changing the operating status of the member to ERROR. A valid value is from 1 to 10.
name	string	Human-readable name of the monitor.
pool_id	string	The pool to monitor.
timeout	int	The time, in seconds, after which a health check times out. This value must be less than the delay value.
type	string	The type of health monitor. One of HTTP, HTTPS, PING, TCP, or TLS-HELLO.
url_path	string	The HTTP URL path of the request sent by the monitor to test the health of a backend member. Must be a string that begins with a forward slash (/).

- **delete**

  Deletes an existing health monitor.

  Octavia will pass in the health monitor object as a parameter.

  The health monitor will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

- **update**

  Modifies an existing health monitor using the values supplied in the health monitor object.

  Octavia will pass in the original health monitor object which is the baseline for the update, and a health monitor object with the fields to be updated.

  As of the writing of this specification the update health monitor object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
delay	int	The interval, in seconds, between health checks.
expected_codes	string	The expected HTTP status codes to get from a successful health check. This may be a single value, a list, or a range.
healthmonitor_id	string	ID of health monitor to create.
http_method	string	The HTTP method that the health monitor uses for requests. One of CONNECT, DELETE, GET, HEAD, OPTIONS, PATCH, POST, PUT, or TRACE.
max_retries	int	The number of successful checks before changing the operating status of the member to ONLINE.
max_retries_down	int	The number of allowed check failures before changing the operating status of the member to ERROR. A valid value is from 1 to 10.
name	string	Human-readable name of the monitor.
timeout	int	The time, in seconds, after which a health check times out. This value must be less than the delay value.
url_path	string	The HTTP URL path of the request sent by the monitor to test the health of a backend member. Must be a string that begins with a forward slash (/).

The health monitor will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the health monitor to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def health_monitor_create(self, healthmonitor):
      """Creates a new health monitor.

      :param healthmonitor (object): The health monitor object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def health_monitor_delete(self, healthmonitor):
      """Deletes a healthmonitor_id.

      :param healthmonitor (object): The health monitor object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
```

(continues on next page)

(continued from previous page)

```
        """
        raise NotImplementedError()

    def health_monitor_update(self, old_healthmonitor, new_healthmonitor):
        """Updates a health monitor.

        :param old_healthmonitor (object): The baseline health monitor
          object.
        :param new_healthmonitor (object): The updated health monitor object.
        :return: Nothing if the create request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()
```

L7 Policy

- **create**

  Creates an L7 policy.

  Octavia will pass in the L7 policy object with all requested settings.

  The L7 policy will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the L7 policy to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The l7policy python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The project_id will be removed, if present, as this field is now deprecated. The l7policy will inherit the project_id from the parent load balancer.

  L7policy object

  As of the writing of this specification the create l7policy object may contain the following:

Name	Type	Description
action	string	The L7 policy action. One of REDIRECT_TO_POOL, REDIRECT_TO_URL, or REJECT.
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the L7 policy.
l7policy_id	string	The ID of the L7 policy.
listener_id	string	The ID of the listener.
name	string	Human-readable name of the L7 policy.
position	int	The position of this policy on the listener. Positions start at 1.
redirect_pool_id	string	Requests matching this policy will be redirected to the pool with this ID. Only valid if action is REDIRECT_TO_POOL.
redirect_url	string	Requests matching this policy will be redirected to this URL. Only valid if action is REDIRECT_TO_URL.
rules	list	A list of l7rule objects.

Creating a Fully Populated L7 policy

If the "rules" option is specified, the provider driver will create all of the child objects in addition to creating the L7 policy instance.

- **delete**

  Deletes an existing L7 policy.

  Octavia will pass in the L7 policy object as a parameter.

  The l7policy will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

- **update**

  Modifies an existing L7 policy using the values supplied in the l7policy object.

  Octavia will pass in the original L7 policy object which is the baseline for the update, and an L7 policy object with the fields to be updated.

  As of the writing of this specification the update L7 policy object may contain the following:

Name	Type	Description
action	string	The L7 policy action. One of REDIRECT_TO_POOL, REDIRECT_TO_URL, or REJECT.
admin_state_up	bool	Admin state: True if up, False if down.
description	string	A human-readable description for the L7 policy.
l7policy_id	string	The ID of the L7 policy.
name	string	Human-readable name of the L7 policy.
position	int	The position of this policy on the listener. Positions start at 1.
redirect_pool_id	string	Requests matching this policy will be redirected to the pool with this ID. Only valid if action is REDIRECT_TO_POOL.
redirect_url	string	Requests matching this policy will be redirected to this URL. Only valid if action is REDIRECT_TO_URL.

The L7 policy will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the L7 policy to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
  def l7policy_create(self, l7policy):
      """Creates a new L7 policy.

      :param l7policy (object): The l7policy object.
      :return: Nothing if the create request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      :raises UnsupportedOptionError: if driver does not
        support one of the configuration options.
      """
      raise NotImplementedError()

  def l7policy_delete(self, l7policy):
      """Deletes an L7 policy.

      :param l7policy (object): The l7policy object.
      :return: Nothing if the delete request was accepted.
      :raises DriverError: An unexpected error occurred in the driver.
      :raises NotImplementedError: if driver does not support request.
      """
      raise NotImplementedError()

 def l7policy_update(self, old_l7policy, new_l7policy):
      """Updates an L7 policy.

      :param old_l7policy (object): The baseline l7policy object.
```

(continues on next page)

(continued from previous page)

```
        :param new_l7policy (object): The updated l7policy object.
        :return: Nothing if the update request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()
```

L7 Rule

- **create**

  Creates a new L7 rule for an existing L7 policy.

  Octavia will pass in the L7 rule object with all requested settings.

  The L7 rule will be in the `PENDING_CREATE` provisioning_status and `OFFLINE` operating_status when it is passed to the driver. The driver will be responsible for updating the provisioning status of the L7 rule to either `ACTIVE` if successfully created, or `ERROR` if not created.

  The Octavia API will accept and do basic API validation of the create request from the user. The l7rule python object representing the request body will be passed to the driver create method as it was received and validated with the following exceptions:

  1. The project_id will be removed, if present, as this field is now deprecated. The listener will inherit the project_id from the parent load balancer.

  L7rule object

  As of the writing of this specification the create l7rule object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
compare_type	string	The comparison type for the L7 rule. One of CONTAINS, ENDS_WITH, EQUAL_TO, REGEX, or STARTS_WITH.
invert	bool	When True the logic of the rule is inverted. For example, with invert True, equal to would become not equal to.
key	string	The key to use for the comparison. For example, the name of the cookie to evaluate.
l7policy_id	string	The ID of the L7 policy.
l7rule_id	string	The ID of the L7 rule.
type	string	The L7 rule type. One of COOKIE, FILE_TYPE, HEADER, HOST_NAME, or PATH.
value	string	The value to use for the comparison. For example, the file type to compare.

- **delete**

  Deletes an existing L7 rule.

  Octavia will pass in the L7 rule object as a parameter.

The L7 rule will be in the `PENDING_DELETE` provisioning_status when it is passed to the driver. The driver will notify Octavia that the delete was successful by setting the provisioning_status to `DELETED`. If the delete failed, the driver will update the provisioning_status to `ERROR`.

- **update**

  Modifies an existing L7 rule using the values supplied in the l7rule object.

  Octavia will pass in the original L7 rule object which is the baseline for the update, and an L7 rule object with the fields to be updated.

  As of the writing of this specification the update L7 rule object may contain the following:

Name	Type	Description
admin_state_up	bool	Admin state: True if up, False if down.
compare_type	string	The comparison type for the L7 rule. One of CONTAINS, ENDS_WITH, EQUAL_TO, REGEX, or STARTS_WITH.
invert	bool	When True the logic of the rule is inverted. For example, with invert True, equal to would become not equal to.
key	string	The key to use for the comparison. For example, the name of the cookie to evaluate.
l7rule_id	string	The ID of the L7 rule.
type	string	The L7 rule type. One of COOKIE, FILE_TYPE, HEADER, HOST_NAME, or PATH.
value	string	The value to use for the comparison. For example, the file type to compare.

  The L7 rule will be in the `PENDING_UPDATE` provisioning_status when it is passed to the driver. The driver will update the provisioning_status of the L7 rule to either `ACTIVE` if successfully updated, or `ERROR` if the update was not successful.

  The driver is expected to validate that the driver supports the request. The method will then return or raise an exception if the request cannot be accepted.

Abstract class definition

```
class Driver(object):
    def l7rule_create(self, l7rule):

        """Creates a new L7 rule.

        :param l7rule (object): The L7 rule object.
        :return: Nothing if the create request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()

    def l7rule_delete(self, l7rule):

        """Deletes an L7 rule.
```

(continues on next page)

(continued from previous page)

```
        :param l7rule (object): The L7 rule object.
        :return: Nothing if the delete request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        """
        raise NotImplementedError()

    def l7rule_update(self, old_l7rule, new_l7rule):

        """Updates an L7 rule.

        :param old_l7rule (object): The baseline L7 rule object.
        :param new_l7rule (object): The updated L7 rule object.
        :return: Nothing if the update request was accepted.
        :raises DriverError: An unexpected error occurred in the driver.
        :raises NotImplementedError: if driver does not support request.
        :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
        """
        raise NotImplementedError()
```

Flavor

Octavia flavors are defined in a separate specification (see References below). Support for flavors will be provided through two provider driver interfaces, one to query supported flavor metadata keys and another to validate that a flavor is supported. Both functions are synchronous.

- **get_supported_flavor_keys**

  Retrieves a dictionary of supported flavor keys and their description.

  ```
  {"topology": "The load balancer topology for the flavor. One of: SINGLE,␣
  ↪ACTIVE_STANDBY",
   "compute_flavor": "The compute driver flavor to use for the load␣
  ↪balancer instances"}
  ```

- **validate_flavor**

  Validates that the driver supports the flavor metadata dictionary.

  The validate_flavor method will be passed a flavor metadata dictionary that the driver will validate. This is used when an operator uploads a new flavor that applies to the driver.

  The validate_flavor method will either return or raise a `UnsupportedOptionError` exception.

Following are interface definitions for flavor support:

```
def get_supported_flavor_metadata():
    """Returns a dictionary of flavor metadata keys supported by this driver.
```

(continues on next page)

(continued from previous page)

```
    The returned dictionary will include key/value pairs, 'name' and
    'description.'

    :returns: The flavor metadata dictionary
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: The driver does not support flavors.
    """
    raise NotImplementedError()
```

```
def validate_flavor(flavor_metadata):
    """Validates if driver can support flavor as defined in flavor_metadata.

    :param flavor_metadata (dict): Dictionary with flavor metadata.
    :return: Nothing if the flavor is valid and supported.
    :raises DriverError: An unexpected error occurred in the driver.
    :raises NotImplementedError: The driver does not support flavors.
    :raises UnsupportedOptionError: if driver does not
          support one of the configuration options.
    """
    raise NotImplementedError()
```

Exception Model

DriverError

This is a catch all exception that drivers can return if there is an unexpected error. An example might be a delete call for a load balancer the driver does not recognize. This exception includes two strings: The user fault string and the optional operator fault string. The user fault string, "user_fault_string", will be provided to the API requester. The operator fault string, "operator_fault_string", will be logged in the Octavia API log file for the operator to use when debugging.

```
class DriverError(Exception):
    user_fault_string = _("An unknown driver error occurred.")
    operator_fault_string = _("An unknown driver error occurred.")

    def __init__(self, *args, **kwargs):
        self.user_fault_string = kwargs.pop('user_fault_string',
                                            self.user_fault_string)
        self.operator_fault_string = kwargs.pop('operator_fault_string',
                                                self.operator_fault_string)

        super(DriverError, self).__init__(*args, **kwargs)
```

NotImplementedError

Driver implementations may not support all operations, and are free to reject a request. If the driver does not implement an API function, the driver will raise a NotImplementedError exception.

```
class NotImplementedError(Exception):
    user_fault_string = _("A feature is not implemented by this driver.")
    operator_fault_string = _("A feature is not implemented by this driver.")

    def __init__(self, *args, **kwargs):
        self.user_fault_string = kwargs.pop('user_fault_string',
                                            self.user_fault_string)
        self.operator_fault_string = kwargs.pop('operator_fault_string',
                                                self.operator_fault_string)

        super(NotImplementedError, self).__init__(*args, **kwargs)
```

UnsupportedOptionError

Provider drivers will validate that they can complete the request -- that all options are supported by the driver. If the request fails validation, drivers will raise an UnsupportedOptionError exception. For example, if a driver does not support a flavor passed as an option to load balancer create(), the driver will raise an UnsupportedOptionError and include a message parameter providing an explanation of the failure.

```
class UnsupportedOptionError(Exception):
    user_fault_string = _("A specified option is not supported by this driver.
↪")
    operator_fault_string = _("A specified option is not supported by this␣
↪driver.")

    def __init__(self, *args, **kwargs):
        self.user_fault_string = kwargs.pop('user_fault_string',
                                            self.user_fault_string)
        self.operator_fault_string = kwargs.pop('operator_fault_string',
                                                self.operator_fault_string)

        super(UnsupportedOptionError, self).__init__(*args, **kwargs)
```

Driver Support Library

Provider drivers need support for updating provisioning status, operating status, and statistics. Drivers will not directly use database operations, and instead will callback to Octavia using a new API.

> **Warning:** The methods listed here are the only callable methods for drivers. All other interfaces are not considered stable or safe for drivers to access.

Update provisioning and operating status API

The update status API defined below can be used by provider drivers to update the provisioning and/or operating status of Octavia resources (load balancer, listener, pool, member, health monitor, L7 policy, or L7 rule).

For the following status API, valid values for provisioning status and operating status parameters are as defined by Octavia status codes. If an existing object is not included in the input parameter, the status remains unchanged.

provisioning_status: status associated with lifecycle of the resource. See Octavia Provisioning Status Codes.

operating_status: the observed status of the resource. See Octavia Operating Status Codes.

The dictionary takes this form:

```
{ "loadbalancers": [{"id": "123",
                     "provisioning_status": "ACTIVE",
                     "operating_status": "ONLINE"},...],
  "healthmonitors": [],
  "l7policies": [],
  "l7rules": [],
  "listeners": [],
  "members": [],
  "pools": []
}
```

```
def update_loadbalancer_status(status):
    """Update load balancer status.

    :param status (dict): dictionary defining the provisioning status and
        operating status for load balancer objects, including pools,
        members, listeners, L7 policies, and L7 rules.
    :raises: UpdateStatusError
    :returns: None
    """
```

Update statistics API

Provider drivers can update statistics for listeners using the following API. Similar to the status function above, a single dictionary with multiple listener statistics is used to update statistics in a single call. If an existing listener is not included, the statistics for that object will remain unchanged.

The general form of the input dictionary is a list of listener statistics:

```
{ "listeners": [{"id": "123",
                 "active_connections": 12,
                 "bytes_in": 238908,
                 "bytes_out": 290234,
                 "request_errors": 0,
```

(continues on next page)

(continued from previous page)

```
                                "total_connections": 3530},...]
}
```

```
def update_listener_statistics(statistics):
    """Update listener statistics.

    :param statistics (dict): Statistics for listeners:
          id (string): ID of the listener.
          active_connections (int): Number of currently active connections.
          bytes_in (int): Total bytes received.
          bytes_out (int): Total bytes sent.
          request_errors (int): Total requests not fulfilled.
          total_connections (int): The total connections handled.
    :raises: UpdateStatisticsError
    :returns: None
    """
```

Get Resource Support

Provider drivers may need to get information about an Octavia resource. As an example of its use, a provider driver may need to sync with Octavia, and therefore need to fetch all of the Octavia resources it is responsible for managing. Provider drivers can use the existing Octavia API to get these resources. See the Octavia API Reference.

API Exception Model

The driver support API will include two Exceptions, one for each of the two API groups:

- UpdateStatusError
- UpdateStatisticsError

Each exception class will include a message field that describes the error and references to the failed record if available.

```
class UpdateStatusError(Exception):
    fault_string = _("The status update had an unknown error.")
    status_object = None
    status_object_id = None
    status_record = None

    def __init__(self, *args, **kwargs):
        self.fault_string = kwargs.pop('fault_string',
                                       self.fault_string)
        self.status_object = kwargs.pop('status_object', None)
        self.status_object_id = kwargs.pop('status_object_id', None)
        self.status_record = kwargs.pop('status_record', None)

```

(continues on next page)

(continued from previous page)

```
        super(UnsupportedOptionError, self).__init__(*args, **kwargs)

class UpdateStatisticsError(Exception):
    fault_string = _("The statistics update had an unknown error.")
    stats_object = None
    stats_object_id = None
    stats_record = None

    def __init__(self, *args, **kwargs):
        self.fault_string = kwargs.pop('fault_string',
                                       self.fault_string)
        self.stats_object = kwargs.pop('stats_object', None)
        self.stats_object_id = kwargs.pop('stats_object_id', None)
        self.stats_record = kwargs.pop('stats_record', None)

        super(UnsupportedOptionError, self).__init__(*args, **kwargs)
```

Alternatives

Driver Support Library

An alternative to this library is a REST interface that drivers use directly. A REST implementation can still be used within the library, but wrapping it in an API simplifies the programming interface.

Data model impact

None, the required data model changes are already present.

REST API impact

None, the required REST API changes are already present.

Security impact

None.

Notifications impact

None.

Other end user impact

Users will be able to direct requests to specific backends using the *provider* parameter. Users may want to understand the availability of provider drivers, and can use Octavia APIs to do so.

Performance Impact

The performance impact on Octavia should be minimal. Driver requests will need to be scheduled, and Octavia will process driver callbacks through a REST interface. As provider drivers are loaded by Octavia, calls into drivers are through direct interfaces.

Other deployer impact

Minimal configuration is needed to support provider drivers. The work required is adding a driver name to Octavia's configuration file, and installing provider drivers supplied by third parties.

Developer impact

The proposal defines interaction between Octavia and backend drivers, so no developer impact is expected.

Implementation

Assignee(s)

Work Items

- Implement loading drivers defined the Octavia configuration.
- Implement scheduling requests to drivers.
- Implement validating flavors with provider drivers.
- Implement getting and testing flavors with provider drivers.
- Implement a no-op driver for testing.
- Implement driver support library functions:
  - Update status functions
  - Update statistics functions
- Migrate the existing Octavia reference driver to use this interface.

Dependencies

- Octavia API: https://docs.openstack.org/api-ref/load-balancer/
- Flavors: https://docs.openstack.org/octavia/latest/contributor/specs/version1.0/flavors.html

Testing

Tempest tests should be added for testing:

- Scheduling: test that Octavia effectively schedules to drivers besides the default driver.
- Request validation: test request validation API.
- Flavor profile validation: test flavor validation.
- Flavor queries: test flavor queries.
- Statistics updates

Functional API tests should be updated to test the provider API.

Documentation Impact

A driver developer guide should be created.

References

Octavia API https://docs.openstack.org/api-ref/load-balancer/v2/index.html

Octavia Flavors Specification https://docs.openstack.org/octavia/latest/contributor/specs/version1.0/flavors.html

UDP Support

https://storyboard.openstack.org/#!/story/1657091

Problem description

Currently, the default driver of Octavia (haproxy) only supports TCP, HTTP, HTTPS, and TERMINATED_HTTPS. We need support for load balancing UDP.

For some use-cases, UDP load balancing support is useful. One such case are real-time media streaming applications which are based on RTSP[1].

For the Internet of Things (IoT)[2], there are many services or applications that use UDP as their transmission protocol. For example: CoAP[3] (Constrained Application Protocol), DDS[4] (Data Distribution

---

[1] https://en.wikipedia.org/wiki/Real_Time_Streaming_Protocol

[2] https://en.wikipedia.org/wiki/Internet_of_things

[3] https://en.wikipedia.org/wiki/Constrained_Application_Protocol

[4] https://en.wikipedia.org/wiki/Data_Distribution_Service

Service) for Real-Time systems, and the introduction protocol Thread[5].

Applications with high demand for real-time (like video chatting) run on RDUP[6] (Reliable User Datagram Protocol), RTP[7] (RealTime Protocol) and UDT[8] (UDP-based Data Transfer Protocol). These protocols also are based on UDP.

There isn't any option in the API for these protocols, which Layer 4 UDP would provide. This means that customers lack a way to support these services which may be running on VM instances in an OpenStack environment.

Proposed change

This spec extends the LBaaSv2 API to support *UDP* as a protocol in Listener and Pool resource requests.

It will require a new load balancing engine to support this feature, as the current haproxy engine only supports TCP based protocols. If users want a load balancer which supports both TCP and UDP, this need cannot be met by launching haproxy-based amphora instances. It's the good time to extend octavia to support more load balancing scenarios. This spec will introduce how LVS[9] can work with haproxy for UDP loadbalancing. The reason for choosing LVS is that we can easily integrate it with the existing `keepalived` service. That means we can configure LVS via `keepalived`, and check member health as well.

For the current service VM driver implementation, haproxy runs in the amphora-haproxy namespace in an amphora instance. So we also need to configure `keeplived` in the same namespace for UDP cases even in SINGLE topology. For ACTIVE_STANDBY, `keepalived` will serve two purposes: UDP and VRRP. So, one instance of `keepalived` must be bound in the namespace, along with the LVS instance it configures.

The main idea is to use `keepalived` to configure and manage LVS[10] and its configuration. We also need to check the members' statuses with `keepalived` instead of `haproxy`, so there must be a different workflow in Octavia resources and deployment topologies. The simplest implementation is LVS within NAT mode, so we will only support this mode to start. If possible we will add other modes in the future.

Currently, a single `keepalived` instance can support multiple virtual server configurations, but for minimal impact of reconfiguration to the existing listeners, we'd better not to refresh all the `keepalived` configuration files and restart the instances, because that would cause all listeners traffic to be blocked if the LVS configuration maintained by `keepalived` is removed. This spec proposes that each listener will have its own `keepalived` process, but that process won't contain a VRRP instance, just the configuration of virtual server and real servers. That means if the Loadbalancer service is running with ACTIVE-STANDBY topology, each amphora instance will run multiple `keepalived` instances, the count being N+1 (where N is the UDP `Listener` count, and +1 is the VRRP instance for HA). The existing `keepalived` will be used, but each "UDP Listener keepalived process" will need to be controlled by health check of the Main VRRP keepalived process. Then the VIP could be moved to the BACKUP amphorae instance in ACTIVE/STANDBY topology if there is any issue with these UDP keepalived processes. The health check will simply reflect whether the keepalived processes are alive.

The workflow for this feature contains:

1. Add a new `keepalived` jinja template to support LVS configuration.

---

[5] https://en.wikipedia.org/wiki/Thread_(network_protocol)

[6] https://en.wikipedia.org/wiki/Reliable_User_Datagram_Protocol

[7] https://de.wikipedia.org/wiki/Real-Time_Transport_Protocol

[8] https://en.wikipedia.org/wiki/UDP-based_Data_Transfer_Protocol

[9] http://www.linuxvirtualserver.org/

[10] https://github.com/acassen/keepalived/blob/master/doc/keepalived.conf.SYNOPSIS#L559

2. Add `netcat` into dib-elements for supporting all platforms.
3. Extend the ability of amphora agent to run `keepalived` with LVS configuration in amphora instances, including the init configuration, such as systemd, sysvinit and upstart.
4. Enhance the session persistence to work with UDP and enable/disable the "One-Packet-Scheduling" option.
5. Update the database to allow listeners to support both `tcp` and `udp` on the same port, add `udp` as a valid protocol and `ONE_PACKET_SCHEDULING` as a valid session_persistence_type in the database.
6. Setup validation code for supported features of UDP load balancing (such as session persistence, types of health monitors, load balancing algorithms, number of L7 policies allowed, etc).
7. Extend the existing LBaaSv2 API in Octavia to allow `udp` parameters in the `Listener` resource.
8. Extend the Loadbalancer/Listener flows to support udp loadbalancer in the particular topologies.

Alternatives

Introduce a new UDP driver based on LVS or other Loadbalancer engines. Then find a way to fix the gap of the current Octavia data models which have a strong relationship with HTTP which based on TCP.

Provide a new driver provider framework to change the amphorae backend from haproxy to some other load balancer engines, for example, if we introduce LVS driver, we may just support the simple L7 functions with LVS, as it's a risk to change provider from existing haproxy-based amphora instances to LVS ones. If possible, we need to limit the API to not support fields/resources if the backend driver is LVS, such as "insert_headers" in Listener, L7Policies, L7Rules and etc, a series fields/resources that related to L7 layer. The all things are to match the real ability of backend. That means all the configuration of L7 resources will be ignored or translate to LVS configuration if the backend is LVS. For other load balancer engines which support UDP, such as f5/nginx, we may also need to do this.

Combining the 2 load balancer engines for a simple reference implementation, LVS would only support the L4 layer LB, and haproxy would provide the L7 LB functionality which is more specific and detailed. For other engines like f5/nginx, Octavia can directly pass the UDP parameters to backend. This is very good for the community environment. Then Octavia may support more powerful and complex LoadBalancing solutions.

Data model impact

There may not be any data model changes, this spec just allows a user to input the `udp` protocol to create/update the `Listener` and `Pool` resources. So here, just extend the `SUPPORTED_PROTOCOLS` to add the value `PROTOCOL_UDP`.

```
SUPPORTED_PROTOCOLS = (PROTOCOL_TCP, PROTOCOL_HTTPS, PROTOCOL_HTTP,
                       PROTOCOL_TERMINATED_HTTPS, PROTOCOL_PROXY,
                       PROTOCOL_UDP)
```

Also add a record into the table `protocol` for `PROTOCOL_UDP`.

As LVS only operates in Layer 4, there are some conflicts with current Octavia data models. There are some limitation below:

1. No L7 policies allowed.

2. For session persistence, this spec will intro `persistence_timeout` (sec) and `persistence_granularity` (subnet mask)[11] in the virtual server configuration. The function will be based on the LVS. With no session persistence specified, LVS will be configured with a persistence_timeout of 0. There are two valid session persistence options for UDP (if session persistence is specified), `SOURCE_IP` and `ONE_PACKET_SCHEDULING`.
3. Intro a 'UDP_CONNECT' type for UDP in `healthmonitor`, for the simple, only check the UDP port is open by `nc` command. And for current API of `healthmonitor`, we need to make clear the meaning of LVS with the current `healthmonitor` API like the mapping below

Option Mapping Healthmonitor->LVS	Healthmonitor Description	Keepalived LVS Description
delay -> delay_loop	Set the time in seconds, between sending probes to members.	Delay timer for service polling.
max_retires_down -> retry	Set the number of allowed check failure before changing the operating status of the member to ERROR.	Number of retries before fail.
timeout -> delay_before_retry	Set the maximum time, in seconds, that a monitor waits to connect before it times out. This value must be less than the delay value.	delay before retry (default 1 unless otherwise specified)

4. For UDP load balancing, we can support the same algorithms at first. Such as SOURCE_IP(sh), ROUND_ROBIN(rr) and LEAST_CONNECTIONS(lc).

REST API impact

- Allow the `protocol` fields to accept `udp`.
- Allow the `healthmonitor.type` field to accept UDP type values.
- Add some fields to `session_persistence` that are specific to UDP though `SOURCE_IP` type and a new type `ONE_PACKET_SCHEDULING`.

Create/Update Listener Request:

```
POST/PUT /v2.0/lbaas/listeners
{
    "listener": {
        "admin_state_up": true,
        "connection_limit": 100,
        "description": "listener one",
        "loadbalancer_id": "a36c20d0-18e9-42ce-88fd-82a35977ee8c",
        "name": "listener1",
        "protocol": "UDP",
        "protocol_port": "18000"
    }
}
```

---

[11] http://www.linuxvirtualserver.org/docs/persistence.html

Note: It is the same as the current relationships, where one `listener` will have only one default `pool` for UDP. A `loadbalancer` can have multiple `listeners` for UDP loadbalancing on different ports.

Create/Update Pool Request

`SOURCE_IP` type case:

```
POST/PUT /v2.0/lbaas/pools

{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "lb_algorithm": "ROUND_ROBIN",
        "name": "my-pool",
        "protocol": "UDP",
        "session_persistence": {
            "type": "SOURCE_IP",
            "persistence_timeout": 60,
            "persistence_granularity": "255.255.0.0",
        }
        "listener_id": "39de4d56-d663-46e5-85a1-5b9d5fa17829",
    }
}
```

`ONE_PACKET_SCHEDULING` type case:

```
POST/PUT /v2.0/lbaas/pools

{
    "pool": {
        "admin_state_up": true,
        "description": "simple pool",
        "lb_algorithm": "ROUND_ROBIN",
        "name": "my-pool",
        "protocol": "UDP",
        "session_persistence": {
            "type": "ONE_PACKET_SCHEDULING"
        }
        "listener_id": "39de4d56-d663-46e5-85a1-5b9d5fa17829",
    }
}
```

Note: The validation part for UDP will just allow to set the specific fields which associated with UDP. For example, user can not set the `protocol` with "udp" and `insert_headers` in the same request.

Create/Update Health Monitor Request:

```
POST/PUT /v2.0/lbaas/healthmonitors

{
    "healthmonitor": {
        "name": "Good health monitor"
        "admin_state_up": true,
        "pool_id": "c5e9e801-0473-463b-a017-90c8e5237bb3",
        "delay": 10,
        "max_retries": 4,
        "max_retries_down": 4,
        "timeout": 5,
        "type": "UDP_CONNECT"
    }
}
```

Note: We don't allow to create a `healthmonitor` with any other L7 parameters, like "http_method", "url_path" and "expected_code" if the associated `pool` support UDP. But for the positional option "max_retries", it's different from API description in keepalived/LVS, so the default value is the same as the value of "max_retires_down" if user specified. In general, "max_retires_down" should be overridden by "max_retries".

Security impact

The security should be affected by the UDP server, we need to add another neutron security group rule to the existing security group to support UDP. Security impact is minimal as the keepalived/LVS will be running in the tenant traffic network namespace.

Notifications impact

No expected change.

Other end user impact

Users will be able to pass "UDP" to create/update Listener/Pool resources for UDP load balancer.

Performance Impact

- If enabled driver is LVS, it will have a good performance for L4 load balancing, but lack the any functionality in L7.
- As this spec introduces LVS and Haproxy working together, if users update the `Listener` or `Pool` resources in a `LoadBalancer` instance frequently, the loadbalancer functionality may be delayed for a while as the refresh of UDP related LVS configuration.
- As we need to add keepalived monitoring process for each UDP listeners, it is necessary to consider RAM about amphora VM instances.

Other deployer impact

No expected change.

Developer impact

No expected change.

Implementation

Assignee(s)

zhaobo

Work Items

- Add/extend startup script templates for keepalived processes, including configuration.
- Extend the ability of existing amphora agent and driver to generate and control LVS by `keepalived` in amphora instances.
- Extend the exist Octavia V2 API to access `udp` parameter in `Listener` and `pools` resources.
- Extend the Loadbalancer/Listener flows to support udp loadbalancer in the particular topologies.
- Extend Octavia V2 API to accept UDP fields.
- Add the specified logic which involved into haproxy agent and the affected resource workflow in Octavia.
- Add API validation code to validate the fields of UDP cases.
- Add Unit Tests to Octavia.
- Add API functional tests.
- Add scenario tests into octavia tempest plugin.
- Update CLI and Octavia-dashboard to support UDP fields input.
- Documentation work.

Dependencies

None

Testing

Unit tests, Functional tests, API tests and Scenario tests are necessary.

Documentation Impact

The description of Octavia API reference will need to be updated. The load balancing cookbook should be also updated. Make it clear the difference of `healthmonitor` behaviors in UDP cases.

References

4.6 Module Reference

4.6.1 octavia

octavia package

Subpackages

octavia.amphorae package

Subpackages

octavia.amphorae.backends package

Subpackages

octavia.amphorae.backends.agent package

Subpackages

octavia.amphorae.backends.agent.api_server package

Submodules

octavia.amphorae.backends.agent.api_server.amphora_info module

`class` **`AmphoraInfo`**`(`*osutils*`)`

Bases: `object`

`compile_amphora_details``(`*extend_lvs_driver=None*`)`

`compile_amphora_info``(`*extend_lvs_driver=None*`)`

`get_interface``(`*ip_addr*`)`

octavia.amphorae.backends.agent.api_server.certificate_update module

upload_server_cert()

octavia.amphorae.backends.agent.api_server.haproxy_compatibility module

get_haproxy_versions()

Get major and minor version number from haproxy

Returns major_version The major version digit

Returns minor_version The minor version digit

process_cfg_for_version_compat(*haproxy_cfg*)

octavia.amphorae.backends.agent.api_server.keepalived module

class Keepalived

Bases: `object`

manager_keepalived_service(*action*)

upload_keepalived_config()

octavia.amphorae.backends.agent.api_server.keepalivedlvs module

class KeepalivedLvs

Bases: *octavia.amphorae.backends.agent.api_server.lvs_listener_base.LvsListenerApiServerBase*

delete_lvs_listener(*listener_id*)

Delete a LVS Listener from a amphora

Parameters **listener_id** -- The id of the listener

Returns HTTP response with status code.

Raises **Exception** -- If unsupport initial system of amphora.

get_all_lvs_listeners_status()

Gets the status of all UDP listeners

Gets the status of all UDP listeners on the amphora.

get_lvs_listener_config(*listener_id*)

Gets the keepalivedlvs config

Parameters **listener_id** -- the id of the listener

manage_lvs_listener(*listener_id*, *action*)

Gets the LVS Listener configuration details

Parameters

- **listener_id** -- the id of the LVS Listener
- **action** -- the operation type.

Returns HTTP response with status code.

Raises **`Exception`** -- If the listener is failed to find.

`upload_lvs_listener_config`(*listener_id*)

Upload the configuration for LVS.

Parameters **`listener_id`** -- The id of a LVS Listener

Returns HTTP response with status code.

Raises **`Exception`** -- If any file / directory is not found or fail to create.

octavia.amphorae.backends.agent.api_server.loadbalancer module

`class` **`Loadbalancer`**

Bases: `object`

`delete_certificate`(*lb_id*, *filename*)

`delete_lb`(*lb_id*)

`get_all_listeners_status`(*other_listeners=None*)

Gets the status of all listeners

This method will not consult the stats socket so a listener might show as ACTIVE but still be in ERROR

Currently type==SSL is also not detected

`get_certificate_md5`(*lb_id*, *filename*)

`get_haproxy_config`(*lb_id*)

Gets the haproxy config

Parameters **`listener_id`** -- the id of the listener

`start_stop_lb`(*lb_id*, *action*)

`upload_certificate`(*lb_id*, *filename*)

`upload_haproxy_config`(*amphora_id*, *lb_id*)

Upload the haproxy config

Parameters

- **`amphora_id`** -- The id of the amphora to update
- **`lb_id`** -- The id of the loadbalancer

`class` **`Wrapped`**(*stream_*)

Bases: `object`

`get_md5`()

`read`(*line*)

octavia.amphorae.backends.agent.api_server.lvs_listener_base module

`class LvsListenerApiServerBase`

Bases: `object`

Base LVS Listener Server API

`abstract delete_lvs_listener(`*listener_id*`)`

Delete a LVS Listener from a amphora

Parameters `listener_id` -- The id of the listener

Returns HTTP response with status code.

Raises `Exception` -- If unsupport initial system of amphora.

`abstract get_all_lvs_listeners_status()`

Gets the status of all LVS Listeners

This method will not consult the stats socket so a listener might show as ACTIVE but still be in ERROR

Returns a list of LVS Listener status

Raises `Exception` -- If the listener pid located directory is not exist

`abstract get_lvs_listener_config(`*listener_id*`)`

Gets the LVS Listener configuration details

Parameters `listener_id` -- the id of the LVS Listener

Returns HTTP response with status code.

Raises `Exception` -- If the listener is failed to find.

`get_subscribed_amp_compile_info()`

`abstract manage_lvs_listener(`*listener_id*, *action*`)`

Gets the LVS Listener configuration details

Parameters

- `listener_id` -- the id of the LVS Listener
- `action` -- the operation type.

Returns HTTP response with status code.

Raises `Exception` -- If the listener is failed to find.

`abstract upload_lvs_listener_config(`*listener_id*`)`

Upload the configuration for LVS.

Parameters `listener_id` -- The id of a LVS Listener

Returns HTTP response with status code.

Raises `Exception` -- If any file / directory is not found or fail to create.

octavia.amphorae.backends.agent.api_server.osutils module

class BaseOS(*os_name*)

Bases: `object`

classmethod bring_interface_up(*interface*, *name*)

classmethod get_os_util()

write_interface_file(*interface*, *ip_address*, *prefixlen*)

write_port_interface_file(*interface*, *fixed_ips*, *mtu*)

write_vip_interface_file(*interface*, *vip*, *ip_version*, *prefixlen*, *gateway*, *mtu*, *vrrp_ip*, *host_routes*, *fixed_ips=None*)

class CentOS(*os_name*)

Bases: `octavia.amphorae.backends.agent.api_server.osutils.RH`

classmethod is_os_name(*os_name*)

class RH(*os_name*)

Bases: `octavia.amphorae.backends.agent.api_server.osutils.BaseOS`

cmd_get_version_of_installed_package(*package_name*)

classmethod is_os_name(*os_name*)

class Ubuntu(*os_name*)

Bases: `octavia.amphorae.backends.agent.api_server.osutils.BaseOS`

cmd_get_version_of_installed_package(*package_name*)

classmethod is_os_name(*os_name*)

octavia.amphorae.backends.agent.api_server.plug module

class Plug(*osutils*)

Bases: `object`

plug_lo()

plug_network(*mac_address*, *fixed_ips*, *mtu=None*, *vip_net_info=None*)

plug_vip(*vip*, *subnet_cidr*, *gateway*, *mac_address*, *mtu=None*, *vrrp_ip=None*, *host_routes=None*)

octavia.amphorae.backends.agent.api_server.server module

class Server

Bases: `object`

delete_certificate(*lb_id*, *filename*)

delete_lb_object(*object_id*)

get_all_listeners_status()

get_certificate_md5(*lb_id*, *filename*)

get_details()

get_haproxy_config(*lb_id*)

get_info()

get_interface(*ip_addr*)

get_lvs_listener_config(*listener_id*)

manage_service_vrrp(*action*)

plug_network()

plug_vip(*vip*)

start_stop_lb_object(*object_id*, *action*)

upload_cert()

upload_certificate(*lb_id*, *filename*)

upload_config()

upload_haproxy_config(*amphora_id*, *lb_id*)

upload_lvs_listener_config(*amphora_id*, *listener_id*)

upload_vrrp_config()

version_discovery()

make_json_error(*ex*)

register_app_error_handler(*app*)

octavia.amphorae.backends.agent.api_server.util module

exception ParsingError

Bases: `Exception`

exception UnknownInitError

Bases: `Exception`

config_path(*lb_id*)

get_backend_for_lb_object(*object_id*)

Returns the backend for a listener.

If the listener is a TCP based listener return 'HAPROXY'. If the listener is a UDP or SCTP based listener return 'LVS' If the listener is not identifiable, return None.

Parameters **`listener_id`** -- The ID of the listener to identify.

Returns HAPROXY_BACKEND, LVS_BACKEND or None

get_haproxy_pid(*lb_id*)

get_haproxy_vip_addresses(*lb_id*)

Get the VIP addresses for a load balancer.

Parameters **`lb_id`** -- The load balancer ID to get VIP addresses from.

Returns List of VIP addresses (IPv4 and IPv6)

get_keepalivedlvs_pid(*listener_id*)

get_listeners()

Get Listeners

Returns An array with the ids of all listeners, e.g. ['123', '456', ...] or [] if no listeners exist

get_loadbalancers()

Get Load balancers

Returns An array with the uuids of all load balancers, e.g. ['123', '456', ...] or [] if no loadbalancers exist

get_lvs_listeners()

get_os_init_system()

haproxy_check_script_path()

haproxy_dir(*lb_id*)

haproxy_sock_path(*lb_id*)

init_path(*lb_id*, *init_system*)

install_netns_systemd_service()

is_lb_running(*lb_id*)

is_lvs_listener_running(*listener_id*)

keepalived_backend_check_script_dir()

keepalived_backend_check_script_path()

keepalived_cfg_path()

keepalived_check_script_path()

keepalived_check_scripts_dir()

keepalived_dir()

keepalived_init_path(*init_system*)

keepalived_log_path()

keepalived_lvs_cfg_path(*listener_id*)

keepalived_lvs_dir()

keepalived_lvs_init_path(*init_system*, *listener_id*)

keepalived_lvs_pids_path(*listener_id*)

keepalived_pid_path()

parse_haproxy_file(*lb_id*)

pid_path(*lb_id*)

run_systemctl_command(*command*, *service*)

send_member_advertisements(*fixed_ips: Iterable[Dict[str, str]]*)

Sends advertisements for each fixed_ip of a list

This method will send either GARP (IPv4) or neighbor advertisements (IPv6) for the addresses of the subnets of the members.

Parameters **fixed_ips** -- a list of dicts that contain 'ip_address' elements

Returns None

send_vip_advertisements(*lb_id*)

Sends address advertisements for each load balancer VIP.

This method will send either GARP (IPv4) or neighbor advertisements (IPv6) for the VIP addresses on a load balancer.

Parameters **lb_id** -- The load balancer ID to send advertisements for.

Returns None

state_file_path(*lb_id*)

vrrp_check_script_update(*lb_id*, *action*)

Module contents

Submodules

octavia.amphorae.backends.agent.agent_jinja_cfg module

class AgentJinjaTemplater

Bases: `object`

build_agent_config(*amphora_id*, *topology*)

Module contents

octavia.amphorae.backends.health_daemon package

Submodules

octavia.amphorae.backends.health_daemon.health_daemon module

build_stats_message()

Build a stats message based on retrieved listener statistics.

Example version 3 message without UDP (note that values are deltas, not absolutes):

```
{"id": "<amphora_id>",
 "seq": 67,
 "listeners": {
   "<listener_id>": {
     "status": "OPEN",
```

(continues on next page)

(continued from previous page)

```
          "stats": {
            "tx": 0,
            "rx": 0,
            "conns": 0,
            "totconns": 0,
            "ereq": 0
          }
        }
      },
      "pools": {
          "<pool_id>:<listener_id>": {
            "status": "UP",
            "members": {
              "<member_id>": "no check"
            }
          }
      },
      "ver": 3
    }
```

calculate_stats_deltas(*listener_id*, *row*)

get_counters()

get_counters_file()

get_stats(*stat_sock_file*)

list_sock_stat_files(*hadir=None*)

persist_counters()

Attempt to persist the latest statistics values

run_sender(*cmd_queue*)

octavia.amphorae.backends.health_daemon.health_sender module

class UDPStatusSender

Bases: `object`

dosend(*obj*)

update(*dest*, *port*)

round_robin_addr(*addrinfo_list*)

octavia.amphorae.backends.health_daemon.status_message module

`decode_obj`(*binary_array*)

`encode_obj`(*obj*)

`get_hmac`(*payload*, *key*, *hex=True*)

Get digest for the payload.

The hex param is for backward compatibility, so the package data sent from the existing amphorae can still be checked in the previous approach.

`get_payload`(*envelope*, *key*, *hex=True*)

`to_hex`(*byte_array*)

`unwrap_envelope`(*envelope*, *key*)

A backward-compatible way to get data.

We may still receive package from amphorae that are using digest() instead of hexdigest()

`wrap_envelope`(*obj*, *key*, *hex=True*)

Module contents

octavia.amphorae.backends.utils package

Submodules

octavia.amphorae.backends.utils.haproxy_query module

`class HAProxyQuery`(*stats_socket*)

Bases: `object`

Class used for querying the HAProxy statistics socket.

The CSV output is defined in the HAProxy documentation:

http://cbonte.github.io/haproxy-dconv/configuration-1.4.html#9

`get_pool_status`()

Get status for each server and the pool as a whole.

Returns pool data structure {<pool-name>: { 'uuid': <uuid>, 'status': 'UP'|'DOWN', 'members': [<name>: 'UP'|'DOWN'|'DRAIN'|'no check'] }}

`save_state`(*state_file_path*)

Save haproxy connection state to a file.

Parameters **`state_file_path`** -- Absolute path to the state file

Returns bool (True if success, False otherwise)

`show_info`()

Get and parse output from 'show info' command.

`show_stat`(*proxy_iid=- 1*, *object_type=- 1*, *server_id=- 1*)

Get and parse output from 'show stat' command.

Parameters

- **proxy_iid** -- Proxy ID (column 27 in CSV output). -1 for all.
- **object_type** -- Select the type of dumpable object. Values can be ORed. -1 - everything 1 - frontends 2 - backends 4 - servers
- **server_id** -- Server ID (column 28 in CSV output?), or -1 for everything.

Returns stats (split into an array by newline)

octavia.amphorae.backends.utils.interface module

class InterfaceController

Bases: `object`

ADD = 'add'

DELETE = 'delete'

FLUSH = 'flush'

SET = 'set'

TENTATIVE_WAIT_INTERVAL = 0.2

TENTATIVE_WAIT_TIMEOUT = 30

down(*interface*)

interface_file_list()

list()

up(*interface*)

octavia.amphorae.backends.utils.interface_file module

class InterfaceFile(*name*, *if_type*, *mtu=None*, *addresses=None*, *routes=None*, *rules=None*, *scripts=None*)

Bases: `object`

classmethod dump(*obj*)

classmethod from_file(*filename*)

classmethod get_directory()

classmethod get_extensions()

classmethod get_host_routes(*routes*, ***kwargs*)

classmethod load(*fp*)

write()

class PortInterfaceFile(*name*, *mtu*, *fixed_ips*)

Bases: `octavia.amphorae.backends.utils.interface_file.InterfaceFile`

class VIPInterfaceFile(*name*, *mtu*, *vip*, *ip_version*, *prefixlen*, *gateway*, *vrrp_ip*, *host_routes*, *topology*, *fixed_ips=None*)

Bases: `octavia.amphorae.backends.utils.interface_file.InterfaceFile`

octavia.amphorae.backends.utils.ip_advertisement module

calculate_icmpv6_checksum(*packet*)

Calculate the ICMPv6 checksum for a packet.

Parameters **packet** -- The packet bytes to checksum.

Returns The checksum integer.

garp(*interface*, *ip_address*, *net_ns=None*)

Sends a gratuitous ARP for ip_address on the interface.

Parameters

- **interface** -- The interface name to send the GARP on.
- **ip_address** -- The IP address to advertise in the GARP.
- **net_ns** -- The network namespace to send the GARP from.

Returns None

neighbor_advertisement(*interface*, *ip_address*, *net_ns=None*)

Sends a unsolicited neighbor advertisement for an ip on the interface.

Parameters

- **interface** -- The interface name to send the GARP on.
- **ip_address** -- The IP address to advertise in the GARP.
- **net_ns** -- The network namespace to send the GARP from.

Returns None

send_ip_advertisement(*interface*, *ip_address*, *net_ns=None*)

Send an address advertisement.

This method will send either GARP (IPv4) or neighbor advertisements (IPv6) for the ip address specified.

Parameters

- **interface** -- The interface name to send the advertisement on.
- **ip_address** -- The IP address to advertise.
- **net_ns** -- The network namespace to send the advertisement from.

Returns None

octavia.amphorae.backends.utils.keepalivedlvs_query module

get_ipvsadm_info(*ns_name*, *is_stats_cmd=False*)

get_listener_realserver_mapping(*ns_name*, *listener_ip_port*, *health_monitor_enabled*)

get_lvs_listener_pool_status(*listener_id*)

get_lvs_listener_resource_ipports_nsname(*listener_id*)

get_lvs_listeners_stats()

read_kernel_file(*ns_name*, *file_path*)

octavia.amphorae.backends.utils.network_namespace module

class NetworkNamespace(*netns*)

Bases: `object`

A network namespace context manager.

Runs wrapped code inside the specified network namespace.

Parameters **netns** -- The network namespace name to enter.

CLONE_NEWNET = 1073741824

octavia.amphorae.backends.utils.network_utils module

get_interface_name(*ip_address*, *net_ns=None*)

Gets the interface name from an IP address.

Parameters

- **ip_address** -- The IP address to lookup.
- **net_ns** -- The network namespace to find the interface in.

Returns The interface name.

Raises

- *exceptions.InvalidIPAddress* -- Invalid IP address provided.
- *octavia.common.exceptions.NotFound* -- No interface was found.

Module contents

Module contents

octavia.amphorae.driver_exceptions package

Submodules

octavia.amphorae.driver_exceptions.exceptions module

exception AmpConnectionRetry(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'Could not connect to amphora, exception caught: %(exception)s'

exception AmpDriverNotImplementedError(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'Amphora does not implement this feature.'

exception AmphoraDriverError(***kwargs*)

Bases: `Exception`

message = 'A super class for all other exceptions and the catch.'

static use_fatal_exceptions()

Return True if use fatal exceptions by raising them.

exception ArchiveException(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = "couldn't archive the logs"

exception DeleteFailed(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = "this load balancer couldn't be deleted"

exception EnableFailed(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = "this load balancer couldn't be enabled"

exception HealthMonitorProvisioningError(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.ProvisioningErrors*

message = "couldn't provision HealthMonitor"

exception InfoException(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'gathering information about this amphora failed'

exception ListenerProvisioningError(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.ProvisioningErrors*

message = "couldn't provision Listener"

exception LoadBalancerProvisoningError(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.ProvisioningErrors*

message = "couldn't provision LoadBalancer"

exception MetricsException(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'gathering metrics failed'

exception NodeProvisioningError(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.ProvisioningErrors*

message = "couldn't provision Node"

exception NotFoundError(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = "this amphora couldn't be found"

exception ProvisioningErrors(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'Super class for provisioning amphora errors'

exception StatisticsException(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'gathering statistics failed'

exception SuspendFailed(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = "this load balancer couldn't be suspended"

exception TimeOutException(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = 'contacting the amphora timed out'

exception UnauthorizedException(***kwargs*)

Bases: *octavia.amphorae.driver_exceptions.exceptions.AmphoraDriverError*

message = "the driver can't access the amphora"

Module contents

octavia.amphorae.drivers package

Subpackages

octavia.amphorae.drivers.haproxy package

Submodules

octavia.amphorae.drivers.haproxy.data_models module

class CPU(*total=None*, *user=None*, *system=None*, *soft_irq=None*)

Bases: `octavia.common.data_models.`

class Details(*hostname=None*, *uuid=None*, *version=None*, *api_version=None*, *network_tx=None*, *network_rx=None*, *active=None*, *haproxy_count=None*, *cpu=None*, *memory=None*, *disk=None*, *load=None*, *listeners=None*, *packages=None*)

Bases: `octavia.common.data_models.`

class Disk(*used=None*, *available=None*)

Bases: `octavia.common.data_models.`

class Info(*hostname=None*, *uuid=None*, *version=None*, *api_version=None*)
Bases: `octavia.common.data_models.`

class ListenerStatus(*status=None*, *uuid=None*, *provisioning_status=None*, *type=None*, *pools=None*)
Bases: `octavia.common.data_models.`

class Memory(*total=None*, *free=None*, *available=None*, *buffers=None*, *cached=None*, *swap_used=None*, *shared=None*, *slab=None*, *committed_as=None*)
Bases: `octavia.common.data_models.`

class Pool(*uuid=None*, *status=None*, *members=None*)
Bases: `octavia.common.data_models.`

class Topology(*hostname=None*, *uuid=None*, *topology=None*, *role=None*, *ip=None*, *ha_ip=None*)
Bases: `octavia.common.data_models.`

octavia.amphorae.drivers.haproxy.exceptions module

exception APIException(***kwargs*)
Bases: `webob.exc.HTTPClientError`

`code = 500`

`msg = 'Something unknown went wrong'`

exception Conflict(***kwargs*)
Bases: *octavia.amphorae.drivers.haproxy.exceptions.APIException*

`code = 409`

`msg = 'Conflict'`

exception Forbidden(***kwargs*)
Bases: *octavia.amphorae.drivers.haproxy.exceptions.APIException*

`code = 403`

`msg = 'Forbidden'`

exception InternalServerError(***kwargs*)
Bases: *octavia.amphorae.drivers.haproxy.exceptions.APIException*

`code = 500`

`msg = 'Internal Server Error'`

exception InvalidRequest(***kwargs*)
Bases: *octavia.amphorae.drivers.haproxy.exceptions.APIException*

`code = 400`

`msg = 'Invalid request'`

exception NotFound(***kwargs*)
Bases: *octavia.amphorae.drivers.haproxy.exceptions.APIException*

`code = 404`

`msg = 'Not Found'`

exception **ServiceUnavailable**(***kwargs*)

Bases: `octavia.amphorae.drivers.haproxy.exceptions.APIException`

code = 503

msg = 'Service Unavailable'

exception **Unauthorized**(***kwargs*)

Bases: `octavia.amphorae.drivers.haproxy.exceptions.APIException`

code = 401

msg = 'Unauthorized'

check_exception(*response*, *ignore=()*, *log_error=True*)

octavia.amphorae.drivers.haproxy.rest_api_driver module

class **AmphoraAPIClient0_5**

Bases: `octavia.amphorae.drivers.haproxy.rest_api_driver.AmphoraAPIClientBase`

delete_cert_pem(*amp*, *listener_id*, *pem_filename*)

delete_listener(*amp*, *listener_id*)

get_all_listeners(*amp*)

get_cert_md5sum(*amp*, *listener_id*, *pem_filename*, *ignore=()*)

get_details(*amp*)

get_info(*amp*, *raise_retry_exception=False*, *timeout_dict=None*)

get_interface(*amp*, *ip_addr*, *timeout_dict=None*, *log_error=True*)

plug_network(*amp*, *port*)

plug_vip(*amp*, *vip*, *net_info*)

update_agent_config(*amp*, *agent_config*, *timeout_dict=None*)

update_cert_for_rotation(*amp*, *pem_file*)

upload_cert_pem(*amp*, *listener_id*, *pem_filename*, *pem_file*)

upload_config(*amp*, *listener_id*, *config*, *timeout_dict=None*)

upload_udp_config(*amp*, *listener_id*, *config*, *timeout_dict=None*)

upload_vrrp_config(*amp*, *config*)

class **AmphoraAPIClient1_0**

Bases: `octavia.amphorae.drivers.haproxy.rest_api_driver.AmphoraAPIClientBase`

delete_cert_pem(*amp*, *loadbalancer_id*, *pem_filename*)

delete_listener(*amp*, *object_id*)

get_all_listeners(*amp*)

get_cert_md5sum(*amp*, *loadbalancer_id*, *pem_filename*, *ignore=()*)

get_details(*amp*)

get_info(*amp*, *raise_retry_exception=False*, *timeout_dict=None*)

get_interface(*amp*, *ip_addr*, *timeout_dict=None*, *log_error=True*)

get_listener_status(*amp*, *listener_id*)

plug_network(*amp*, *port*)

plug_vip(*amp*, *vip*, *net_info*)

update_agent_config(*amp*, *agent_config*, *timeout_dict=None*)

update_cert_for_rotation(*amp*, *pem_file*)

upload_cert_pem(*amp*, *loadbalancer_id*, *pem_filename*, *pem_file*)

upload_config(*amp*, *loadbalancer_id*, *config*, *timeout_dict=None*)

upload_udp_config(*amp*, *listener_id*, *config*, *timeout_dict=None*)

upload_vrrp_config(*amp*, *config*)

class **AmphoraAPIClientBase**

Bases: `object`

get_api_version(*amp*, *timeout_dict=None*, *raise_retry_exception=False*)

request(*method*, *amp*, *path='/'*, *timeout_dict=None*, *retry_404=True*, *raise_retry_exception=False*, ***kwargs*)

class **CustomHostNameCheckingAdapter**(*pool_connections=10*, *pool_maxsize=10*, *max_retries=0*, *pool_block=False*)

Bases: `requests.adapters.HTTPAdapter`

cert_verify(*conn*, *url*, *verify*, *cert*)

Verify a SSL certificate. This method should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **conn** -- The urllib3 connection object associated with the cert.
- **url** -- The requested URL.
- **verify** -- Either a boolean, in which case it controls whether we verify the server's TLS certificate, or a string, in which case it must be a path to a CA bundle to use
- **cert** -- The SSL certificate to verify.

init_poolmanager(**pool_args*, ***pool_kwargs*)

Initializes a urllib3 PoolManager.

This method should not be called from user code, and is only exposed for use when subclassing the `HTTPAdapter`.

Parameters

- **connections** -- The number of urllib3 connection pools to cache.
- **maxsize** -- The maximum number of connections to save in the pool.
- **block** -- Block when no free connections are available.

- **`pool_kwargs`** -- Extra keyword arguments used to initialize the Pool Manager.

class `HaproxyAmphoraLoadBalancerDriver`

Bases: *`octavia.amphorae.drivers.driver_base.AmphoraLoadBalancerDriver`*, *`octavia.amphorae.drivers.keepalived.vrrp_rest_driver.KeepalivedAmphoraDriverMixin`*

`check`(*amphora: octavia.db.models.Amphora*, *timeout_dict: Optional[dict] = None*)

Check connectivity to the amphora.

`delete`(*listener*)

Delete the listener on the vip.

Parameters **`listener`** (`octavia.db.models.Listener`) -- listener object, need to use its protocol_port property

Returns return a value list (listener, vip, status flag--delete)

At this moment, we just build the basic structure for testing, will add more function along with the development.

`finalize_amphora`(*amphora*)

Finalize the amphora before any listeners are configured.

Parameters **`amphora`** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property

Returns None

At this moment, we just build the basic structure for testing, will add more function along with the development. This is a hook for drivers who need to do additional work before an amphora becomes ready to accept listeners. Please keep in mind that amphora might be kept in an offline pool after this call.

`get_diagnostics`(*amphora*)

Return ceilometer ready diagnostic data.

Parameters **`amphora`** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property

Returns return a value list (amphora.id, status flag--'ge t_diagnostics')

At this moment, we just build the basic structure for testing, will add more function along with the development, eventually, we want it run some expensive self tests to determine if the amphora and the lbs are healthy the idea is that those tests are triggered more infrequent than the health gathering.

`get_info`(*amphora*, *raise_retry_exception=False*, *timeout_dict=None*)

Returns information about the amphora.

Parameters

- **`amphora`** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property
- **`raise_retry_exception`** -- Flag if outside task should be retried

Returns return a value list (amphora.id, status flag--'info')

At this moment, we just build the basic structure for testing, will add more function along with the development, eventually, we want it to return information as: {"Rest Interface": "1.0", "Amphorae": "1.0", "packages":{"ha proxy":"1.5"}} some information might come from querying the amphora

get_interface_from_ip(*amphora*, *ip_address*, *timeout_dict=None*)

Get the interface name for an IP address.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- The amphora to query.
- **ip_address** (*string*) -- The IP address to lookup. (IPv4 or IPv6)
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns the interface name string if found.

Raises

- *octavia.amphorae.drivers.haproxy.exceptions.NotFound* -- No interface found on the amphora
- *TimeOutException* -- The amphora didn't reply

post_network_plug(*amphora*, *port*, *amphora_network_config*)

Called after amphora added to network

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **port** (`octavia.network.data_models.Port`) -- contains information of the plugged port
- **amphora_network_config** (*octavia.network.data_models.AmphoraNetworkConfig*) -- A data model containing information about the subnets and ports that an amphorae owns.

This method is optional to implement. After adding an amphora to a network, there may be steps necessary on the amphora to allow it to access said network. Ex: creating an interface on an amphora for a neutron network to utilize.

post_vip_plug(*amphora*, *load_balancer*, *amphorae_network_config*, *vrrp_port=None*, *vip_subnet=None*)

Called after network driver has allocated and plugged the VIP

Parameters

- **amphora** (`octavia.db.models.Amphora`) --
- **load_balancer** (`octavia.common.data_models.LoadBalancer`) -- A load balancer that just had its vip allocated and plugged in the network driver.
- **amphorae_network_config** (*octavia.network.data_models.AmphoraNetworkConfig*) -- A data model containing information about the subnets and ports that an amphorae owns.

- **vrrp_port** (`octavia.network.data_models.Port`) -- VRRP port associated with the load balancer
- **vip_subnet** (`octavia.network.data_models.Subnet`) -- VIP subnet associated with the load balancer

Returns None

This is to do any additional work needed on the amphorae to plug the vip, such as bring up interfaces.

reload(*loadbalancer*, *amphora=None*, *timeout_dict=None*)

Reload the listeners on the amphora.

Parameters

- **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object to reload listeners
- **amphora** (`octavia.db.models.Amphora`) -- Amphora to start. If None, reload on all amphora
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns return a value list (listener, vip, status flag--enable)

At this moment, we just build the basic structure for testing, will add more function along with the development.

start(*loadbalancer*, *amphora=None*, *timeout_dict=None*)

Start the listeners on the amphora.

Parameters

- **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object to start listeners
- **amphora** (`octavia.db.models.Amphora`) -- Amphora to start. If None, start on all amphora
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns return a value list (listener, vip, status flag--enable)

At this moment, we just build the basic structure for testing, will add more function along with the development.

update(*loadbalancer*)

Update the amphora with a new configuration.

Parameters **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object, need to use its vip.ip_address property

Returns None

At this moment, we just build the basic structure for testing, will add more function along with the development.

`update_amphora_agent_config`(*amphora*, *agent_config*, *timeout_dict=None*)

Update the amphora agent configuration file.

Parameters

- **`amphora`** (*`object`*) -- The amphora to update.
- **`agent_config`** (*`string`*) -- The new amphora agent configuration.
- **`timeout_dict`** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns None

Note: This will mutate the amphora agent config and adopt the new values.

`update_amphora_listeners`(*loadbalancer*, *amphora*, *timeout_dict=None*)

Update the amphora with a new configuration.

Parameters

- **`loadbalancer`** (*`object`*) -- The load balancer to update
- **`amphora`** (*`object`*) -- The amphora to update
- **`timeout_dict`** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns None

Updates the configuration of the listeners on a single amphora.

`upload_cert_amp`(*amp*, *pem*)

Upload cert info to the amphora.

Parameters

- **`amphora`** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **`pem_file`** (*`file object`*) -- a certificate file

Upload cert file to amphora for Controller Communication.

Module contents

octavia.amphorae.drivers.health package

Submodules

octavia.amphorae.drivers.health.heartbeat_udp module

`class UDPStatusGetter`

Bases: `object`

This class defines methods that will gather heartbeats

The heartbeats are transmitted via UDP and this class will bind to a port and absorb them

check()

dorecv(**args*, ***kw*)

Waits for a UDP heart beat to be sent.

Returns Returns the unwrapped payload and addr that sent the heartbeat.

update(*key*, *ip*, *port*)

Update the running config for the udp socket server

Parameters

- **key** -- The hmac key used to verify the UDP packets. String
- **ip** -- The ip address the UDP server will read from
- **port** -- The port the UDP server will read from

Returns None

class UpdateHealthDb

Bases: `object`

update_health(*health*, *srcaddr*)

update_stats(*health_message*)

Parses the health message then passes it to the stats driver(s)

Parameters **health_message** (*dict*) -- The health message containing the listener stats

Example V1 message:

```
health = {
    "id": "<amphora_id>",
    "listeners": {
        "<listener_id>": {
            "status": "OPEN",
            "stats": {
                "ereq": 0,
                "conns": 0,
                "totconns": 0,
                "rx": 0,
                "tx": 0,
            },
            "pools": {
                "<pool_id>": {
                    "status": "UP",
                    "members": {"<member_id>": "ONLINE"}
                }
            }
        }
    }
}
```

Example V2 message:

```
{"id": "<amphora_id>",
 "seq": 67,
 "listeners": {
   "<listener_id>": {
     "status": "OPEN",
     "stats": {
       "tx": 0,
       "rx": 0,
       "conns": 0,
       "totconns": 0,
       "ereq": 0
     }
   }
 },
 "pools": {
     "<pool_id>:<listener_id>": {
       "status": "UP",
       "members": {
         "<member_id>": "no check"
       }
     }
 },
 "ver": 2
 "recv_time": time.time()
}
```

Example V3 message:

```
Same as V2 message, except values are deltas rather than absolutes.
```

Module contents

octavia.amphorae.drivers.keepalived package

Subpackages

octavia.amphorae.drivers.keepalived.jinja package

Submodules

octavia.amphorae.drivers.keepalived.jinja.jinja_cfg module

class KeepalivedJinjaTemplater(*keepalived_template=None*)

Bases: `object`

build_keepalived_config(*loadbalancer*, *amphora*, *vip_cidr*)

Renders the loadblanacer keepalived configuration for Active/Standby

Parameters

- **loadbalancer** -- A lodabalancer object
- **amp** -- An amphora object
- **vip_cidr** -- The VIP subnet cidr

get_template(*template_file*)
Returns the specified Jinja configuration template.

Module contents

Submodules

octavia.amphorae.drivers.keepalived.vrrp_rest_driver module

class KeepalivedAmphoraDriverMixin
Bases: *octavia.amphorae.drivers.driver_base.VRRPDriverMixin*

reload_vrrp_service(*loadbalancer*)
Reload the VRRP services of all amphorae of the loadbalancer

Parameters **loadbalancer** -- loadbalancer object

start_vrrp_service(*amphora*, *timeout_dict=None*)
Start the VRRP services on an amphorae.

Parameters

- **amphora** -- amphora object
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

stop_vrrp_service(*loadbalancer*)
Stop the vrrp services running on the loadbalancer's amphorae

Parameters **loadbalancer** -- loadbalancer object

update_vrrp_conf(*loadbalancer*, *amphorae_network_config*, *amphora*, *timeout_dict=None*)
Update amphora of the loadbalancer with a new VRRP configuration

Parameters

- **loadbalancer** -- loadbalancer object
- **amphorae_network_config** -- amphorae network configurations
- **amphora** -- The amphora object to update.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Module contents

octavia.amphorae.drivers.noop_driver package

Submodules

octavia.amphorae.drivers.noop_driver.driver module

class **NoopAmphoraLoadBalancerDriver**

Bases: *octavia.amphorae.drivers.driver_base.AmphoraLoadBalancerDriver*, *octavia.amphorae.drivers.driver_base.VRRPDriverMixin*

check(*amphora*, *timeout_dict=None*)

Check connectivity to the amphora.

Parameters

- **amphora** -- The amphora to query.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Raises ***TimeOutException*** -- The amphora didn't reply

delete(*listener*)

Delete the listener on the vip.

Parameters **listener** (`octavia.db.models.Listener`) -- listener object, need to use its protocol_port property

Returns return a value list (listener, vip, status flag--delete)

At this moment, we just build the basic structure for testing, will add more function along with the development.

finalize_amphora(*amphora*)

Finalize the amphora before any listeners are configured.

Parameters **amphora** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property

Returns None

At this moment, we just build the basic structure for testing, will add more function along with the development. This is a hook for drivers who need to do additional work before an amphora becomes ready to accept listeners. Please keep in mind that amphora might be kept in an offline pool after this call.

get_diagnostics(*amphora*)

Return ceilometer ready diagnostic data.

Parameters **amphora** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property

Returns return a value list (amphora.id, status flag--'ge t_diagnostics')

At this moment, we just build the basic structure for testing, will add more function along with the development, eventually, we want it run some expensive self tests to determine if the amphora and the lbs are healthy the idea is that those tests are triggered more infrequent than the health gathering.

`get_info`(*amphora*, *raise_retry_exception=False*)

Returns information about the amphora.

Parameters

- **`amphora`** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property
- **`raise_retry_exception`** -- Flag if outside task should be retried

Returns return a value list (amphora.id, status flag--'info')

At this moment, we just build the basic structure for testing, will add more function along with the development, eventually, we want it to return information as: {"Rest Interface": "1.0", "Amphorae": "1.0", "packages":{"ha proxy":"1.5"}} some information might come from querying the amphora

`get_interface_from_ip`(*amphora*, *ip_address*, *timeout_dict=None*)

Get the interface name from an IP address.

Parameters

- **`amphora`** (`octavia.db.models.Amphora`) -- The amphora to query.
- **`ip_address`** (*`string`*) -- The IP address to lookup. (IPv4 or IPv6)
- **`timeout_dict`** (*`dict`*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

`post_network_plug`(*amphora*, *port*, *amphora_network_config*)

Called after amphora added to network

Parameters

- **`amphora`** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **`port`** (`octavia.network.data_models.Port`) -- contains information of the plugged port
- **`amphora_network_config`** (*`octavia.network.data_models.AmphoraNetworkConfig`*) -- A data model containing information about the subnets and ports that an amphorae owns.

This method is optional to implement. After adding an amphora to a network, there may be steps necessary on the amphora to allow it to access said network. Ex: creating an interface on an amphora for a neutron network to utilize.

`post_vip_plug`(*amphora*, *load_balancer*, *amphorae_network_config*, *vrrp_port=None*, *vip_subnet=None*)

Called after network driver has allocated and plugged the VIP

Parameters

- **`amphora`** (`octavia.db.models.Amphora`) --

- **load_balancer** (`octavia.common.data_models.LoadBalancer`) -- A load balancer that just had its vip allocated and plugged in the network driver.
- **amphorae_network_config** (*octavia.network.data_models.AmphoraNetworkConfig*) -- A data model containing information about the subnets and ports that an amphorae owns.
- **vrrp_port** (`octavia.network.data_models.Port`) -- VRRP port associated with the load balancer
- **vip_subnet** (`octavia.network.data_models.Subnet`) -- VIP subnet associated with the load balancer

Returns None

This is to do any additional work needed on the amphorae to plug the vip, such as bring up interfaces.

`reload`(*loadbalancer*, *amphora=None*, *timeout_dict=None*)

Reload the listeners on the amphora.

Parameters

- **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object to reload listeners
- **amphora** (`octavia.db.models.Amphora`) -- Amphora to start. If None, reload on all amphora
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns return a value list (listener, vip, status flag--enable)

At this moment, we just build the basic structure for testing, will add more function along with the development.

`reload_vrrp_service`(*loadbalancer*)

Reload the VRRP services of all amphorae of the loadbalancer

Parameters **loadbalancer** -- loadbalancer object

`start`(*loadbalancer*, *amphora=None*, *timeout_dict=None*)

Start the listeners on the amphora.

Parameters

- **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object to start listeners
- **amphora** (`octavia.db.models.Amphora`) -- Amphora to start. If None, start on all amphora
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns return a value list (listener, vip, status flag--enable)

At this moment, we just build the basic structure for testing, will add more function along with the development.

start_vrrp_service(*amphora*, *timeout_dict=None*)
Start the VRRP services on the amphora

Parameters

- **amphora** -- The amphora object to start the service on.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

stop_vrrp_service(*loadbalancer*)
Stop the vrrp services running on the loadbalancer's amphorae

Parameters **loadbalancer** -- loadbalancer object

update(*loadbalancer*)
Update the amphora with a new configuration.

Parameters **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object, need to use its vip.ip_address property

Returns None

At this moment, we just build the basic structure for testing, will add more function along with the development.

update_amphora_agent_config(*amphora*, *agent_config*)
Upload and update the amphora agent configuration.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **agent_config** (*string*) -- The new amphora agent configuration file.

update_amphora_listeners(*loadbalancer*, *amphora*, *timeout_dict*)
Update the amphora with a new configuration.

Parameters

- **loadbalancer** (*list(*`octavia.db.models.Listener`*)*) -- List of listeners to update.
- **amphora** (`octavia.db.models.Amphora`) -- The index of the specific amphora to update
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns None

Builds a new configuration, pushes it to the amphora, and reloads the listener on one amphora.

update_vrrp_conf(*loadbalancer*, *amphorae_network_config*, *amphora*, *timeout_dict=None*)
Update amphorae of the loadbalancer with a new VRRP configuration

Parameters

- **loadbalancer** -- loadbalancer object

- **amphorae_network_config** -- amphorae network configurations
- **amphora** -- The amphora object to update.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

upload_cert_amp(*amphora*, *pem_file*)

Upload cert info to the amphora.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **pem_file** (*file object*) -- a certificate file

Upload cert file to amphora for Controller Communication.

class **NoopManager**

Bases: `object`

delete(*listener*)

finalize_amphora(*amphora*)

get_diagnostics(*amphora*)

get_info(*amphora*, *raise_retry_exception=False*)

get_interface_from_ip(*amphora*, *ip_address*, *timeout_dict=None*)

post_network_plug(*amphora*, *port*, *amphora_network_config*)

post_vip_plug(*amphora*, *load_balancer*, *amphorae_network_config*, *vrrp_port=None*, *vip_subnet=None*)

reload(*loadbalancer*, *amphora=None*, *timeout_dict=None*)

start(*loadbalancer*, *amphora=None*, *timeout_dict=None*)

update(*loadbalancer*)

update_amphora_agent_config(*amphora*, *agent_config*)

update_amphora_listeners(*loadbalancer*, *amphora*, *timeout_dict*)

upload_cert_amp(*amphora*, *pem_file*)

Module contents

Submodules

octavia.amphorae.drivers.driver_base module

class **AmphoraLoadBalancerDriver**

Bases: `object`

abstract check(*amphora: octavia.db.models.Amphora*, *timeout_dict: Optional[dict] = None*)

Check connectivity to the amphora.

Parameters

- **amphora** -- The amphora to query.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Raises *TimeOutException* -- The amphora didn't reply

abstract delete(*listener*)

Delete the listener on the vip.

Parameters **listener** (`octavia.db.models.Listener`) -- listener object, need to use its protocol_port property

Returns return a value list (listener, vip, status flag--delete)

At this moment, we just build the basic structure for testing, will add more function along with the development.

abstract finalize_amphora(*amphora*)

Finalize the amphora before any listeners are configured.

Parameters **amphora** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property

Returns None

At this moment, we just build the basic structure for testing, will add more function along with the development. This is a hook for drivers who need to do additional work before an amphora becomes ready to accept listeners. Please keep in mind that amphora might be kept in an offline pool after this call.

abstract get_diagnostics(*amphora*)

Return ceilometer ready diagnostic data.

Parameters **amphora** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property

Returns return a value list (amphora.id, status flag--'ge t_diagnostics')

At this moment, we just build the basic structure for testing, will add more function along with the development, eventually, we want it run some expensive self tests to determine if the amphora and the lbs are healthy the idea is that those tests are triggered more infrequent than the health gathering.

abstract get_info(*amphora*, *raise_retry_exception=False*)

Returns information about the amphora.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, need to use its id property
- **raise_retry_exception** -- Flag if outside task should be retried

Returns return a value list (amphora.id, status flag--'info')

At this moment, we just build the basic structure for testing, will add more function along with the development, eventually, we want it to return information as: {"Rest Interface": "1.0", "Amphorae": "1.0", "packages":{"ha proxy":"1.5"}} some information might come from querying the amphora

`abstract get_interface_from_ip`(*amphora*, *ip_address*, *timeout_dict=None*)

Get the interface name from an IP address.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- The amphora to query.
- **ip_address** (*string*) -- The IP address to lookup. (IPv4 or IPv6)
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

`post_network_plug`(*amphora*, *port*, *amphora_network_config*)

Called after amphora added to network

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **port** (`octavia.network.data_models.Port`) -- contains information of the plugged port
- **amphora_network_config** (*octavia.network.data_models.AmphoraNetworkConfig*) -- A data model containing information about the subnets and ports that an amphorae owns.

This method is optional to implement. After adding an amphora to a network, there may be steps necessary on the amphora to allow it to access said network. Ex: creating an interface on an amphora for a neutron network to utilize.

`post_vip_plug`(*amphora*, *load_balancer*, *amphorae_network_config*, *vrrp_port=None*, *vip_subnet=None*)

Called after network driver has allocated and plugged the VIP

Parameters

- **amphora** (`octavia.db.models.Amphora`) --
- **load_balancer** (`octavia.common.data_models.LoadBalancer`) -- A load balancer that just had its vip allocated and plugged in the network driver.
- **amphorae_network_config** (*octavia.network.data_models.AmphoraNetworkConfig*) -- A data model containing information about the subnets and ports that an amphorae owns.
- **vrrp_port** (`octavia.network.data_models.Port`) -- VRRP port associated with the load balancer
- **vip_subnet** (`octavia.network.data_models.Subnet`) -- VIP subnet associated with the load balancer

Returns None

This is to do any additional work needed on the amphorae to plug the vip, such as bring up interfaces.

`abstract reload`(*loadbalancer*, *amphora*, *timeout_dict=None*)

Reload the listeners on the amphora.

Parameters

- **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object to reload listeners
- **amphora** (`octavia.db.models.Amphora`) -- Amphora to start. If None, reload on all amphora
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns return a value list (listener, vip, status flag--enable)

At this moment, we just build the basic structure for testing, will add more function along with the development.

`abstract start`(*loadbalancer*, *amphora*, *timeout_dict=None*)

Start the listeners on the amphora.

Parameters

- **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object to start listeners
- **amphora** (`octavia.db.models.Amphora`) -- Amphora to start. If None, start on all amphora
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns return a value list (listener, vip, status flag--enable)

At this moment, we just build the basic structure for testing, will add more function along with the development.

`abstract update`(*loadbalancer*)

Update the amphora with a new configuration.

Parameters **loadbalancer** (`octavia.db.models.LoadBalancer`) -- loadbalancer object, need to use its vip.ip_address property

Returns None

At this moment, we just build the basic structure for testing, will add more function along with the development.

`update_amphora_agent_config`(*amphora*, *agent_config*)

Upload and update the amphora agent configuration.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)

- **agent_config** (*string*) -- The new amphora agent configuration file.

abstract update_amphora_listeners(*loadbalancer*, *amphora*, *timeout_dict*)

Update the amphora with a new configuration.

Parameters

- **loadbalancer** (*list(*`octavia.db.models.Listener`*)*) -- List of listeners to update.
- **amphora** (`octavia.db.models.Amphora`) -- The index of the specific amphora to update
- **timeout_dict** (*dict*) -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Returns None

Builds a new configuration, pushes it to the amphora, and reloads the listener on one amphora.

upload_cert_amp(*amphora*, *pem_file*)

Upload cert info to the amphora.

Parameters

- **amphora** (`octavia.db.models.Amphora`) -- amphora object, needs id and network ip(s)
- **pem_file** (*file object*) -- a certificate file

Upload cert file to amphora for Controller Communication.

class VRRPDriverMixin

Bases: `object`

Abstract mixin class for VRRP support in loadbalancer amphorae

Usage: To plug VRRP support in another service driver XYZ, use: @plug_mixin(XYZ) class XYZ: ...

abstract reload_vrrp_service(*loadbalancer*)

Reload the VRRP services of all amphorae of the loadbalancer

Parameters **loadbalancer** -- loadbalancer object

abstract start_vrrp_service(*amphora*, *timeout_dict=None*)

Start the VRRP services on the amphora

Parameters

- **amphora** -- The amphora object to start the service on.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

abstract stop_vrrp_service(*loadbalancer*)

Stop the vrrp services running on the loadbalancer's amphorae

Parameters **loadbalancer** -- loadbalancer object

abstract update_vrrp_conf(*loadbalancer*, *amphorae_network_config*, *amphora*, *timeout_dict=None*)

Update amphorae of the loadbalancer with a new VRRP configuration

Parameters

- **loadbalancer** -- loadbalancer object
- **amphorae_network_config** -- amphorae network configurations
- **amphora** -- The amphora object to update.
- **timeout_dict** -- Dictionary of timeout values for calls to the amphora. May contain: req_conn_timeout, req_read_timeout, conn_max_retries, conn_retry_interval

Module contents

Module contents

octavia.api package

Subpackages

octavia.api.common package

Submodules

octavia.api.common.hooks module

class ContextHook

Bases: `pecan.hooks.PecanHook`

Configures a request context and attaches it to the request.

on_route(*state*)

Override this method to create a hook that gets called upon the start of routing.

Parameters **state** -- The Pecan `state` object for the current request.

class QueryParametersHook

Bases: `pecan.hooks.PecanHook`

before(*state*)

Override this method to create a hook that gets called after routing, but before the request gets passed to your controller.

Parameters **state** -- The Pecan `state` object for the current request.

octavia.api.common.pagination module

class PaginationHelper(*params*, *sort_dir='asc'*)

Bases: `object`

Class helping to interact with pagination functionality

Pass this class to *db.repositories* to apply it on query

apply(*query*, *model*, *enforce_valid_params=True*)

Returns a query with sorting / pagination criteria added.

Pagination works by requiring a unique sort_key specified by sort_keys. (If sort_keys is not unique, then we risk looping through values.) We use the last row in the previous page as the pagination 'marker'. So we must return values that follow the passed marker in the order. With a single-valued sort_key, this would be easy: sort_key > X. With a compound-values sort_key, (k1, k2, k3) we must do this to repeat the lexicographical ordering: (k1 > X1) or (k1 == X1 && k2 > X2) or (k1 == X1 && k2 == X2 && k3 > X3) We also have to cope with different sort_directions. Typically, the id of the last row is used as the client-facing pagination marker, then the actual marker object must be fetched from the db and passed in to us as marker. :param query: the query object to which we should add paging/sorting/filtering :param model: the ORM model class :param enforce_valid_params: check for invalid enteries in self.params

Return type sqlalchemy.orm.query.Query

Returns The query with sorting/pagination/filtering added.

octavia.api.common.types module

class AlpnProtocolType

Bases: `wsme.types.UserType`

basetype

alias of `str`

name = 'alpn_protocol'

static validate(*value*)

Validates whether value is a valid ALPN protocol ID.

class BaseMeta(*name*, *bases*, *dct*)

Bases: `wsme.types.BaseMeta`

class BaseType(***kw*)

Bases: `wsme.types.Base`

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

to_dict(*render_unsets=False*)

Converts Octavia WSME type to dictionary.

Parameters **`render_unsets`** -- If True, will convert items that are WSME Unset types to None. If False, does not add the item

`classmethod translate_dict_keys_to_data_model`(*wsme_dict*)
Translate the keys from wsme class type, to data_model.

`classmethod translate_key_to_data_model`(*key*)
Translate the keys from wsme class type, to data_model.

`class CidrType`
Bases: `wsme.types.UserType`

`basetype`
alias of `str`

`name = 'cidr'`

`static validate`(*value*)
Validates whether value is an IPv4 or IPv6 CIDR.

`class IPAddressType`
Bases: `wsme.types.UserType`

`basetype`
alias of `str`

`name = 'ipaddress'`

`static validate`(*value*)
Validates whether value is an IPv4 or IPv6 address.

`class IdOnlyType`(***kw*)
Bases: *`octavia.api.common.types.BaseType`*

`id`
Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class NameOnlyType`(***kw*)
Bases: *`octavia.api.common.types.BaseType`*

`name`
Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PageType(***kw*)

Bases: `octavia.api.common.types.BaseType`

href

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

rel

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class URLPathType

Bases: `wsme.types.UserType`

basetype

alias of `str`

name = 'url_path'

static validate(*value*)

class URLType(*require_scheme=True*)

Bases: `wsme.types.UserType`

basetype

alias of `str`

name = 'url'

validate(*value*)

Module contents

octavia.api.drivers package

Subpackages

octavia.api.drivers.amphora_driver package

Subpackages

octavia.api.drivers.amphora_driver.v1 package

Submodules

octavia.api.drivers.amphora_driver.v1.driver module

class AmphoraProviderDriver

Bases: `octavia_lib.api.drivers.provider_base.ProviderDriver`

create_vip_port(*loadbalancer_id*, *project_id*, *vip_dictionary*)

Creates a port for a load balancer VIP.

If the driver supports creating VIP ports, the driver will create a VIP port with the primary VIP and all additional VIPs added to the port, and return the vip_dictionary populated with the vip_port_id and a list of vip_dictionaries populated with data from the additional VIPs (which are guaranteed to be in the same Network). This might look like: {'port_id': port_id, 'subnet_id': subnet_id_1, 'ip_address': ip1}, [{'subnet_id': subnet_id_2, 'ip_address': ip2}, {...}, {...}] If the driver does not support port creation, the driver will raise a NotImplementedError.

Parameters

- **loadbalancer_id** (*string*) -- ID of loadbalancer.
- **project_id** (*string*) -- The project ID to create the VIP under.

Param vip_dictionary: The VIP dictionary.

Param additional_vip_dicts: A list of additional VIP dictionaries, with subnets guaranteed to be in the same network as the primary vip_dictionary.

Returns VIP dictionary with vip_port_id + a list of additional VIP dictionaries (vip_dict, additional_vip_dicts).

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- The driver does not support creating VIP ports.

get_supported_availability_zone_metadata()

Returns the valid availability zone metadata keys and descriptions.

This extracts the valid availability zone metadata keys and descriptions from the JSON validation schema and returns it as a dictionary.

Returns Dictionary of availability zone metadata keys and descriptions

Raises **DriverError** -- An unexpected error occurred.

get_supported_flavor_metadata()

Returns the valid flavor metadata keys and descriptions.

This extracts the valid flavor metadata keys and descriptions from the JSON validation schema and returns it as a dictionary.

Returns Dictionary of flavor metadata keys and descriptions.

Raises **DriverError** -- An unexpected error occurred.

health_monitor_create(*healthmonitor*)

Creates a new health monitor.

Parameters **healthmonitor** (*object*) -- The health monitor object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

health_monitor_delete(*healthmonitor*)

Deletes a healthmonitor_id.

Parameters **healthmonitor** (*object*) -- The monitor to delete.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

health_monitor_update(*old_healthmonitor*, *new_healthmonitor*)

Updates a health monitor.

Parameters

- **old_healthmonitor** (*object*) -- The baseline health monitor object.

- **`new_healthmonitor`** (*object*) -- The updated health monitor object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7policy_create`(*l7policy*)
Creates a new L7 policy.

Parameters **`l7policy`** (*object*) -- The L7 policy object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7policy_delete`(*l7policy*)
Deletes an L7 policy.

Parameters **`l7policy`** (*object*) -- The L7 policy to delete.

Returns Nothing if the delete request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

`l7policy_update`(*old_l7policy*, *new_l7policy*)
Updates an L7 policy.

Parameters

- **`old_l7policy`** (*object*) -- The baseline L7 policy object.
- **`new_l7policy`** (*object*) -- The updated L7 policy object.

Returns Nothing if the update request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7rule_create`(*l7rule*)
Creates a new L7 rule.

Parameters **`l7rule`** (*object*) -- The L7 rule object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7rule_delete`(*l7rule*)

Deletes an L7 rule.

Parameters **`l7rule`** (*`object`*) -- The L7 rule to delete.

Returns Nothing if the delete request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

`l7rule_update`(*old_l7rule*, *new_l7rule*)

Updates an L7 rule.

Parameters

- **`old_l7rule`** (*`object`*) -- The baseline L7 rule object.
- **`new_l7rule`** (*`object`*) -- The updated L7 rule object.

Returns Nothing if the update request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`listener_create`(*listener*)

Creates a new listener.

Parameters **`listener`** (*`object`*) -- The listener object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`listener_delete`(*listener*)

Deletes a listener.

Parameters **`listener`** (*`object`*) -- The listener to delete.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

listener_update(*old_listener*, *new_listener*)

Updates a listener.

Parameters

- **old_listener** (*object*) -- The baseline listener object.
- **new_listener** (*object*) -- The updated listener object.

Returns Nothing if the update request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

loadbalancer_create(*loadbalancer*)

Creates a new load balancer.

Parameters **loadbalancer** (*object*) -- The load balancer object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- The driver does not support create.
- **UnsupportedOptionError** -- The driver does not support one of the configuration options.

loadbalancer_delete(*loadbalancer*, *cascade=False*)

Deletes a load balancer.

Parameters

- **loadbalancer** (*object*) -- The load balancer to delete.
- **cascade** (*bool*) -- If True, deletes all child objects (listeners, pools, etc.) in addition to the load balancer.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

loadbalancer_failover(*loadbalancer_id*)

Performs a fail over of a load balancer.

Parameters **loadbalancer_id** (*string*) -- ID of the load balancer to failover.

Returns Nothing if the failover request was accepted.

Raises `DriverError` -- An unexpected error occurred in the driver.

Raises NotImplementedError if driver does not support request.

`loadbalancer_update`(*old_loadbalancer*, *new_loadbalancer*)

Updates a load balancer.

Parameters

- **`old_loadbalancer`** (*`object`*) -- The baseline load balancer object.
- **`new_loadbalancer`** (*`object`*) -- The updated load balancer object.

Returns Nothing if the update request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- The driver does not support request.
- **`UnsupportedOptionError`** -- The driver does not support one of the configuration options.

`member_batch_update`(*pool_id*, *members*)

Creates, updates, or deletes a set of pool members.

Parameters

- **`pool_id`** (*`string`*) -- The id of the pool to update.
- **`members`** (*`list`*) -- List of member objects.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`member_create`(*member*)

Creates a new member for a pool.

Parameters **`member`** (*`object`*) -- The member object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`member_delete`(*member*)

Deletes a pool member.

Parameters **`member`** (*`object`*) -- The member to delete.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.

member_update(*old_member*, *new_member*)

Updates a pool member.

Parameters

- **old_member** (*object*) -- The baseline member object.
- **new_member** (*object*) -- The updated member object.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.
- `UnsupportedOptionError` -- if driver does not support one of the configuration options.

pool_create(*pool*)

Creates a new pool.

Parameters **pool** (*object*) -- The pool object.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.
- `UnsupportedOptionError` -- if driver does not support one of the configuration options.

pool_delete(*pool*)

Deletes a pool and its members.

Parameters **pool** (*object*) -- The pool to delete.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.

pool_update(*old_pool*, *new_pool*)

Updates a pool.

Parameters

- **pool** (*object*) -- The baseline pool object.
- **pool** -- The updated pool object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`validate_availability_zone`(*availability_zone_dict*)

Validates availability zone profile data.

This will validate an availability zone profile dataset against the availability zone settings the amphora driver supports.

Parameters **`availability_zone_dict`** (*`dict`*) -- The availability zone dict to validate.

Returns None

Raises

- **`DriverError`** -- An unexpected error occurred.
- **`UnsupportedOptionError`** -- If the driver does not support one of the availability zone settings.

`validate_flavor`(*flavor_dict*)

Validates flavor profile data.

This will validate a flavor profile dataset against the flavor settings the amphora driver supports.

Parameters **`flavor_dict`** -- The flavor dictionary to validate.

Returns None

Raises

- **`DriverError`** -- An unexpected error occurred.
- **`UnsupportedOptionError`** -- If the driver does not support one of the flavor settings.

Module contents

octavia.api.drivers.amphora_driver.v2 package

Submodules

octavia.api.drivers.amphora_driver.v2.driver module

class `AmphoraProviderDriver`

Bases: `octavia_lib.api.drivers.provider_base.ProviderDriver`

`create_vip_port`(*loadbalancer_id*, *project_id*, *vip_dictionary*)

Creates a port for a load balancer VIP.

If the driver supports creating VIP ports, the driver will create a VIP port with the primary VIP and all additional VIPs added to the port, and return the vip_dictionary populated with the vip_port_id and a list of vip_dictionaries populated with data from the additional VIPs (which are guaranteed to be in the same Network). This might look like: {'port_id': port_id, 'subnet_id': subnet_id_1, 'ip_address': ip1}, [{'subnet_id': subnet_id_2, 'ip_address': ip2}, {...}, {...}] If the driver does not support port creation, the driver will raise a NotImplementedError.

Parameters

- `loadbalancer_id` (*string*) -- ID of loadbalancer.
- `project_id` (*string*) -- The project ID to create the VIP under.

Param vip_dictionary: The VIP dictionary.

Param additional_vip_dicts: A list of additional VIP dictionaries, with subnets guaranteed to be in the same network as the primary vip_dictionary.

Returns VIP dictionary with vip_port_id + a list of additional VIP dictionaries (vip_dict, additional_vip_dicts).

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- The driver does not support creating VIP ports.

`get_supported_availability_zone_metadata()`

Returns the valid availability zone metadata keys and descriptions.

This extracts the valid availability zone metadata keys and descriptions from the JSON validation schema and returns it as a dictionary.

Returns Dictionary of availability zone metadata keys and descriptions

Raises `DriverError` -- An unexpected error occurred.

`get_supported_flavor_metadata()`

Returns the valid flavor metadata keys and descriptions.

This extracts the valid flavor metadata keys and descriptions from the JSON validation schema and returns it as a dictionary.

Returns Dictionary of flavor metadata keys and descriptions.

Raises `DriverError` -- An unexpected error occurred.

`health_monitor_create`(*healthmonitor*)

Creates a new health monitor.

Parameters `healthmonitor` (*object*) -- The health monitor object.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.
- `UnsupportedOptionError` -- if driver does not support one of the configuration options.

health_monitor_delete(*healthmonitor*)

Deletes a healthmonitor_id.

Parameters **healthmonitor** (*object*) -- The monitor to delete.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

health_monitor_update(*old_healthmonitor*, *new_healthmonitor*)

Updates a health monitor.

Parameters

- **old_healthmonitor** (*object*) -- The baseline health monitor object.
- **new_healthmonitor** (*object*) -- The updated health monitor object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

l7policy_create(*l7policy*)

Creates a new L7 policy.

Parameters **l7policy** (*object*) -- The L7 policy object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

l7policy_delete(*l7policy*)

Deletes an L7 policy.

Parameters **l7policy** (*object*) -- The L7 policy to delete.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

l7policy_update(*old_l7policy*, *new_l7policy*)

Updates an L7 policy.

Parameters

- **old_l7policy** (*object*) -- The baseline L7 policy object.
- **new_l7policy** (*object*) -- The updated L7 policy object.

Returns Nothing if the update request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

`l7rule_create`(*l7rule*)

Creates a new L7 rule.

Parameters **l7rule** (*object*) -- The L7 rule object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

`l7rule_delete`(*l7rule*)

Deletes an L7 rule.

Parameters **l7rule** (*object*) -- The L7 rule to delete.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

`l7rule_update`(*old_l7rule*, *new_l7rule*)

Updates an L7 rule.

Parameters

- **old_l7rule** (*object*) -- The baseline L7 rule object.
- **new_l7rule** (*object*) -- The updated L7 rule object.

Returns Nothing if the update request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

`listener_create`(*listener*)

Creates a new listener.

Parameters `listener` (*object*) -- The listener object.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.
- `UnsupportedOptionError` -- if driver does not support one of the configuration options.

`listener_delete`(*listener*)

Deletes a listener.

Parameters `listener` (*object*) -- The listener to delete.

Returns Nothing if the delete request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.

`listener_update`(*old_listener*, *new_listener*)

Updates a listener.

Parameters

- `old_listener` (*object*) -- The baseline listener object.
- `new_listener` (*object*) -- The updated listener object.

Returns Nothing if the update request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.
- `UnsupportedOptionError` -- if driver does not support one of the configuration options.

`loadbalancer_create`(*loadbalancer*)

Creates a new load balancer.

Parameters `loadbalancer` (*object*) -- The load balancer object.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- The driver does not support create.
- `UnsupportedOptionError` -- The driver does not support one of the configuration options.

`loadbalancer_delete`(*loadbalancer*, *cascade=False*)

Deletes a load balancer.

Parameters

- **loadbalancer** (*object*) -- The load balancer to delete.
- **cascade** (*bool*) -- If True, deletes all child objects (listeners, pools, etc.) in addition to the load balancer.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

loadbalancer_failover(*loadbalancer_id*)

Performs a fail over of a load balancer.

Parameters **loadbalancer_id** (*string*) -- ID of the load balancer to failover.

Returns Nothing if the failover request was accepted.

Raises **DriverError** -- An unexpected error occurred in the driver.

Raises NotImplementedError if driver does not support request.

loadbalancer_update(*original_load_balancer*, *new_loadbalancer*)

Updates a load balancer.

Parameters

- **old_loadbalancer** (*object*) -- The baseline load balancer object.
- **new_loadbalancer** (*object*) -- The updated load balancer object.

Returns Nothing if the update request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- The driver does not support request.
- **UnsupportedOptionError** -- The driver does not support one of the configuration options.

member_batch_update(*pool_id*, *members*)

Creates, updates, or deletes a set of pool members.

Parameters

- **pool_id** (*string*) -- The id of the pool to update.
- **members** (*list*) -- List of member objects.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

member_create(*member*)

Creates a new member for a pool.

Parameters **`member`** (*`object`*) -- The member object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`member_delete`(*member*)

Deletes a pool member.

Parameters **`member`** (*`object`*) -- The member to delete.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

`member_update`(*old_member*, *new_member*)

Updates a pool member.

Parameters

- **`old_member`** (*`object`*) -- The baseline member object.
- **`new_member`** (*`object`*) -- The updated member object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`pool_create`(*pool*)

Creates a new pool.

Parameters **`pool`** (*`object`*) -- The pool object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`pool_delete`(*pool*)

Deletes a pool and its members.

Parameters **`pool`** (*`object`*) -- The pool to delete.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

`pool_update`(*old_pool*, *new_pool*)

Updates a pool.

Parameters

- **`pool`** (*`object`*) -- The baseline pool object.
- **`pool`** -- The updated pool object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`validate_availability_zone`(*availability_zone_dict*)

Validates availability zone profile data.

This will validate an availability zone profile dataset against the availability zone settings the amphora driver supports.

Parameters **`availability_zone_dict`** (*`dict`*) -- The availability zone dict to validate.

Returns None

Raises

- **`DriverError`** -- An unexpected error occurred.
- **`UnsupportedOptionError`** -- If the driver does not support one of the availability zone settings.

`validate_flavor`(*flavor_dict*)

Validates flavor profile data.

This will validate a flavor profile dataset against the flavor settings the amphora driver supports.

Parameters **`flavor_dict`** -- The flavor dictionary to validate.

Returns None

Raises

- **`DriverError`** -- An unexpected error occurred.
- **`UnsupportedOptionError`** -- If the driver does not support one of the flavor settings.

Module contents

Submodules

octavia.api.drivers.amphora_driver.availability_zone_schema module

octavia.api.drivers.amphora_driver.flavor_schema module

Module contents

octavia.api.drivers.driver_agent package

Submodules

octavia.api.drivers.driver_agent.driver_get module

`process_get`(*get_data*)

octavia.api.drivers.driver_agent.driver_listener module

class ForkingUDSServer(*server_address*, *RequestHandlerClass*, *bind_and_activate=True*)

Bases: `socketserver.ForkingMixIn`, `socketserver.UnixStreamServer`

class GetRequestHandler(*request*, *client_address*, *server*)

Bases: `socketserver.BaseRequestHandler`

handle()

class StatsRequestHandler(*request*, *client_address*, *server*)

Bases: `socketserver.BaseRequestHandler`

handle()

class StatusRequestHandler(*request*, *client_address*, *server*)

Bases: `socketserver.BaseRequestHandler`

handle()

get_listener(*exit_event*)

stats_listener(*exit_event*)

status_listener(*exit_event*)

octavia.api.drivers.driver_agent.driver_updater module

class DriverUpdater(***kwargs*)

Bases: `object`

update_listener_statistics(*statistics*)

Update listener statistics.

Parameters **statistics** (*dict*) -- Statistics for listeners: id (string): ID for listener. active_connections (int): Number of currently active connections. bytes_in (int): Total bytes received. bytes_out (int): Total bytes sent. request_errors (int): Total requests not fulfilled. total_connections (int): The total connections handled.

Raises UpdateStatisticsError

Returns None

update_loadbalancer_status(*status*)

Update load balancer status.

Parameters **status** (*dict*) -- dictionary defining the provisioning status and operating status for load balancer objects, including pools, members, listeners, L7 policies, and L7 rules. iod (string): ID for the object. provisioning_status (string): Provisioning status for the object. operating_status (string): Operating status for the object.

Raises UpdateStatusError

Returns None

Module contents

octavia.api.drivers.noop_driver package

Submodules

octavia.api.drivers.noop_driver.agent module

noop_provider_agent(*exit_event*)

octavia.api.drivers.noop_driver.driver module

class NoopManager

Bases: `object`

create_vip_port(*loadbalancer_id*, *project_id*, *vip_dictionary*)

get_supported_availability_zone_metadata()

get_supported_flavor_metadata()

health_monitor_create(*healthmonitor*)

health_monitor_delete(*healthmonitor*)

health_monitor_update(*old_healthmonitor*, *new_healthmonitor*)

l7policy_create(*l7policy*)

l7policy_delete(*l7policy*)

l7policy_update(*old_l7policy*, *new_l7policy*)

l7rule_create(*l7rule*)

l7rule_delete(*l7rule*)

l7rule_update(*old_l7rule*, *new_l7rule*)

listener_create(*listener*)

listener_delete(*listener*)

listener_update(*old_listener*, *new_listener*)

loadbalancer_create(*loadbalancer*)

loadbalancer_delete(*loadbalancer*, *cascade=False*)

loadbalancer_failover(*loadbalancer_id*)

loadbalancer_update(*old_loadbalancer*, *new_loadbalancer*)

member_batch_update(*pool_id*, *members*)

member_create(*member*)

member_delete(*member*)

member_update(*old_member*, *new_member*)

pool_create(*pool*)

pool_delete(*pool*)

pool_update(*old_pool*, *new_pool*)

validate_availability_zone(*availability_zone_metadata*)

validate_flavor(*flavor_metadata*)

class **NoopProviderDriver**

Bases: `octavia_lib.api.drivers.provider_base.ProviderDriver`

create_vip_port(*loadbalancer_id*, *project_id*, *vip_dictionary*)

Creates a port for a load balancer VIP.

If the driver supports creating VIP ports, the driver will create a VIP port with the primary VIP and all additional VIPs added to the port, and return the vip_dictionary populated with the vip_port_id and a list of vip_dictionaries populated with data from the additional VIPs (which are guaranteed to be in the same Network). This might look like: {'port_id': port_id, 'subnet_id': subnet_id_1, 'ip_address': ip1}, [{'subnet_id': subnet_id_2, 'ip_address': ip2}, {...}, {...}] If the driver does not support port creation, the driver will raise a NotImplementedError.

Parameters

- **loadbalancer_id** (*string*) -- ID of loadbalancer.
- **project_id** (*string*) -- The project ID to create the VIP under.

Param vip_dictionary: The VIP dictionary.

Param additional_vip_dicts: A list of additional VIP dictionaries, with subnets guaranteed to be in the same network as the primary vip_dictionary.

Returns VIP dictionary with vip_port_id + a list of additional VIP dictionaries (vip_dict, additional_vip_dicts).

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- The driver does not support creating VIP ports.

`get_supported_availability_zone_metadata`()

Returns a dict of supported availability zone metadata keys.

The returned dictionary will include key/value pairs, 'name' and 'description.'

Returns The availability zone metadata dictionary

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- The driver does not support AZs.

`get_supported_flavor_metadata`()

Returns a dict of flavor metadata keys supported by this driver.

The returned dictionary will include key/value pairs, 'name' and 'description.'

Returns The flavor metadata dictionary

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- The driver does not support flavors.

`health_monitor_create`(*healthmonitor*)

Creates a new health monitor.

Parameters **healthmonitor** (*object*) -- The health monitor object.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.
- `NotImplementedError` -- if driver does not support request.
- `UnsupportedOptionError` -- if driver does not support one of the configuration options.

`health_monitor_delete`(*healthmonitor*)

Deletes a healthmonitor_id.

Parameters **healthmonitor** (*object*) -- The monitor to delete.

Returns Nothing if the create request was accepted.

Raises

- `DriverError` -- An unexpected error occurred in the driver.

- **`NotImplementedError`** -- if driver does not support request.

`health_monitor_update`(*old_healthmonitor*, *new_healthmonitor*)

Updates a health monitor.

Parameters

- **`old_healthmonitor`** (*`object`*) -- The baseline health monitor object.
- **`new_healthmonitor`** (*`object`*) -- The updated health monitor object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7policy_create`(*l7policy*)

Creates a new L7 policy.

Parameters **`l7policy`** (*`object`*) -- The L7 policy object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7policy_delete`(*l7policy*)

Deletes an L7 policy.

Parameters **`l7policy`** (*`object`*) -- The L7 policy to delete.

Returns Nothing if the delete request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

`l7policy_update`(*old_l7policy*, *new_l7policy*)

Updates an L7 policy.

Parameters

- **`old_l7policy`** (*`object`*) -- The baseline L7 policy object.
- **`new_l7policy`** (*`object`*) -- The updated L7 policy object.

Returns Nothing if the update request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.

  - **`NotImplementedError`** -- if driver does not support request.
  - **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7rule_create`(*l7rule*)

Creates a new L7 rule.

Parameters **`l7rule`** (*`object`*) -- The L7 rule object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`l7rule_delete`(*l7rule*)

Deletes an L7 rule.

Parameters **`l7rule`** (*`object`*) -- The L7 rule to delete.

Returns Nothing if the delete request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

`l7rule_update`(*old_l7rule*, *new_l7rule*)

Updates an L7 rule.

Parameters

- **`old_l7rule`** (*`object`*) -- The baseline L7 rule object.
- **`new_l7rule`** (*`object`*) -- The updated L7 rule object.

Returns Nothing if the update request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`listener_create`(*listener*)

Creates a new listener.

Parameters **`listener`** (*`object`*) -- The listener object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.

- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

listener_delete(*listener*)

Deletes a listener.

Parameters **listener** (*object*) -- The listener to delete.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

listener_update(*old_listener*, *new_listener*)

Updates a listener.

Parameters

- **old_listener** (*object*) -- The baseline listener object.
- **new_listener** (*object*) -- The updated listener object.

Returns Nothing if the update request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

loadbalancer_create(*loadbalancer*)

Creates a new load balancer.

Parameters **loadbalancer** (*object*) -- The load balancer object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- The driver does not support create.
- **UnsupportedOptionError** -- The driver does not support one of the configuration options.

loadbalancer_delete(*loadbalancer*, *cascade=False*)

Deletes a load balancer.

Parameters

- **loadbalancer** (*object*) -- The load balancer to delete.
- **cascade** (*bool*) -- If True, deletes all child objects (listeners, pools, etc.) in addition to the load balancer.

Returns Nothing if the delete request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

loadbalancer_failover(*loadbalancer_id*)

Performs a fail over of a load balancer.

Parameters **loadbalancer_id** (*string*) -- ID of the load balancer to failover.

Returns Nothing if the failover request was accepted.

Raises **DriverError** -- An unexpected error occurred in the driver.

Raises NotImplementedError if driver does not support request.

loadbalancer_update(*old_loadbalancer*, *new_loadbalancer*)

Updates a load balancer.

Parameters

- **old_loadbalancer** (*object*) -- The baseline load balancer object.
- **new_loadbalancer** (*object*) -- The updated load balancer object.

Returns Nothing if the update request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- The driver does not support request.
- **UnsupportedOptionError** -- The driver does not support one of the configuration options.

member_batch_update(*pool_id*, *members*)

Creates, updates, or deletes a set of pool members.

Parameters

- **pool_id** (*string*) -- The id of the pool to update.
- **members** (*list*) -- List of member objects.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

member_create(*member*)

Creates a new member for a pool.

Parameters **member** (*object*) -- The member object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.

- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

member_delete(*member*)

Deletes a pool member.

Parameters **member** (*object*) -- The member to delete.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

member_update(*old_member*, *new_member*)

Updates a pool member.

Parameters

- **old_member** (*object*) -- The baseline member object.
- **new_member** (*object*) -- The updated member object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

pool_create(*pool*)

Creates a new pool.

Parameters **pool** (*object*) -- The pool object.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.
- **UnsupportedOptionError** -- if driver does not support one of the configuration options.

pool_delete(*pool*)

Deletes a pool and its members.

Parameters **pool** (*object*) -- The pool to delete.

Returns Nothing if the create request was accepted.

Raises

- **DriverError** -- An unexpected error occurred in the driver.
- **NotImplementedError** -- if driver does not support request.

`pool_update`(*old_pool*, *new_pool*)
Updates a pool.

Parameters

- **`pool`** (*`object`*) -- The baseline pool object.
- **`pool`** -- The updated pool object.

Returns Nothing if the create request was accepted.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- if driver does not support request.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`validate_availability_zone`(*availability_zone_metadata*)
Validates if driver can support the availability zone.

Parameters **`availability_zone_metadata`** (*`dict`*) -- Dictionary with az metadata.

Returns Nothing if the availability zone is valid and supported.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- The driver does not support availability zones.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.

`validate_flavor`(*flavor_metadata*)
Validates if driver can support the flavor.

Parameters **`flavor_metadata`** (*`dict`*) -- Dictionary with flavor metadata.

Returns Nothing if the flavor is valid and supported.

Raises

- **`DriverError`** -- An unexpected error occurred in the driver.
- **`NotImplementedError`** -- The driver does not support flavors.
- **`UnsupportedOptionError`** -- if driver does not support one of the configuration options.
- **`octavia_lib.api.drivers.exceptions.NotFound`** -- if the driver cannot find a resource.

Module contents

Submodules

octavia.api.drivers.driver_factory module

get_driver(*provider*)

octavia.api.drivers.utils module

call_provider(*provider*, *driver_method*, **args*, ***kwargs*)

Wrap calls to the provider driver to handle driver errors.

This allows Octavia to return user friendly errors when a provider driver has an issue.

Parameters **driver_method** -- Method in the driver to call.

Raises

- *ProviderDriverError* -- Catch all driver error.
- *ProviderNotImplementedError* -- The driver doesn't support this action.
- *ProviderUnsupportedOptionError* -- The driver doesn't support a provided option.

db_HM_to_provider_HM(*db_hm*)

db_l7policies_to_provider_l7policies(*db_l7policies*)

db_l7policy_to_provider_l7policy(*db_l7policy*)

db_l7rule_to_provider_l7rule(*db_l7rule*)

db_l7rules_to_provider_l7rules(*db_l7rules*)

db_listener_to_provider_listener(*db_listener*, *for_delete=False*)

db_listeners_to_provider_dicts_list_of_dicts(*db_listeners*, *for_delete=False*)

db_listeners_to_provider_listeners(*db_listeners*, *for_delete=False*)

db_loadbalancer_to_provider_loadbalancer(*db_loadbalancer*, *for_delete=False*)

db_member_to_provider_member(*db_member*)

db_members_to_provider_members(*db_members*)

db_pool_to_provider_pool(*db_pool*, *for_delete=False*)

db_pools_to_provider_pools(*db_pools*, *for_delete=False*)

hm_dict_to_provider_dict(*hm_dict*)

l7policy_dict_to_provider_dict(*l7policy_dict*)

l7rule_dict_to_provider_dict(*l7rule_dict*)

lb_dict_to_provider_dict(*lb_dict*, *vip=None*, *db_pools=None*, *db_listeners=None*, *for_delete=False*)

listener_dict_to_provider_dict(*listener_dict*, *for_delete=False*)

member_dict_to_provider_dict(*member_dict*)

pool_dict_to_provider_dict(*pool_dict*, *for_delete=False*)

provider_vip_dict_to_vip_obj(*vip_dictionary*)

vip_dict_to_provider_dict(*vip_dict*)

Module contents

octavia.api.v2 package

Subpackages

octavia.api.v2.controllers package

Submodules

octavia.api.v2.controllers.amphora module

class **AmphoraController**(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:amphora:'

delete(*id*)

Deletes an amphora.

get_all(*fields=None*)

Gets all amphorae.

get_one(*id*, *fields=None*)

Gets a single amphora's details.

class **AmphoraStatsController**(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:amphora:'

get()

class **AmphoraUpdateController**(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:amphora:'

put()

Update amphora agent configuration

class **FailoverController**(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:amphora:'

put()
Fails over an amphora

octavia.api.v2.controllers.availability_zone_profiles module

class **AvailabilityZoneProfileController**(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:availability-zone-profile:'

delete(*availability_zone_profile_id*)
Deletes an Availability Zone Profile

get_all(*fields=None*)
Lists all Availability Zone Profiles.

get_one(*id*, *fields=None*)
Gets an Availability Zone Profile's detail.

post(*availability_zone_profile_*)
Creates an Availability Zone Profile.

put(*id*, *availability_zone_profile_*)
Updates an Availability Zone Profile.

octavia.api.v2.controllers.availability_zones module

class **AvailabilityZonesController**(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:availability-zone:'

delete(*availability_zone_name*)
Deletes an Availability Zone

get_all(*fields=None*)
Lists all Availability Zones.

get_one(*name*, *fields=None*)
Gets an Availability Zone's detail.

post(*availability_zone_*)
Creates an Availability Zone.

put(*name*, *availability_zone_*)

octavia.api.v2.controllers.base module

class BaseController(**args*, ***kwargs*)

Bases: `pecan.rest.RestController`

RBAC_TYPE = None

octavia.api.v2.controllers.flavor_profiles module

class FlavorProfileController(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:flavor-profile:'

delete(*flavor_profile_id*)

Deletes a Flavor Profile

get_all(*fields=None*)

Lists all flavor profiles.

get_one(*id*, *fields=None*)

Gets a flavor profile's detail.

post(*flavor_profile_*)

Creates a flavor Profile.

put(*id*, *flavor_profile_*)

Updates a flavor Profile.

octavia.api.v2.controllers.flavors module

class FlavorsController(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:flavor:'

delete(*flavor_id*)

Deletes a Flavor

get_all(*fields=None*)

Lists all flavors.

get_one(*id*, *fields=None*)

Gets a flavor's detail.

post(*flavor_*)

Creates a flavor.

put(*id*, *flavor_*)

octavia.api.v2.controllers.health_monitor module

class HealthMonitorController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:healthmonitor:'

delete(*id*)

Deletes a health monitor.

get_all(*project_id=None*, *fields=None*)

Gets all health monitors.

get_one(*id*, *fields=None*)

Gets a single healthmonitor's details.

post(*health_monitor_*)

Creates a health monitor on a pool.

put(*id*, *health_monitor_*)

Updates a health monitor.

octavia.api.v2.controllers.l7policy module

class L7PolicyController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:l7policy:'

delete(*id*)

Deletes a l7policy.

get(*id*, *fields=None*)

Gets a single l7policy's details.

get_all(*project_id=None*, *fields=None*)

Lists all l7policies of a listener.

post(*l7policy_*)

Creates a l7policy on a listener.

put(*id*, *l7policy_*)

Updates a l7policy.

octavia.api.v2.controllers.l7rule module

class L7RuleController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:l7rule:'

delete(*id*)

Deletes a l7rule.

get(*id*, *fields=None*)

Gets a single l7rule's details.

get_all(*fields=None*)
Lists all l7rules of a l7policy.

post(*rule_*)
Creates a l7rule on an l7policy.

put(*id*, *l7rule_*)
Updates a l7rule.

octavia.api.v2.controllers.listener module

class ListenersController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:listener:'

delete(*id*)
Deletes a listener from a load balancer.

get_all(*project_id=None*, *fields=None*)
Lists all listeners.

get_one(*id*, *fields=None*)
Gets a single listener's details.

post(*listener_*)
Creates a listener on a load balancer.

put(*id*, *listener_*)
Updates a listener on a load balancer.

class StatisticsController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`, `octavia.common.stats.StatsMixin`

RBAC_TYPE = 'os_load-balancer_api:listener:'

get()

octavia.api.v2.controllers.load_balancer module

class FailoverController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.load_balancer.LoadBalancersController`

put(***kwargs*)
Fails over a loadbalancer

class LoadBalancersController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:loadbalancer:'

delete(*id*, *cascade=False*)
Deletes a load balancer.

get_all(*project_id=None*, *fields=None*)
Lists all load balancers.

get_one(*id*, *fields=None*)
Gets a single load balancer's details.

post(*load_balancer*)
Creates a load balancer.

put(*id*, *load_balancer*)
Updates a load balancer.

class StatisticsController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`, `octavia.common.stats.StatsMixin`

RBAC_TYPE = 'os_load-balancer_api:loadbalancer:'

get()

class StatusController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:loadbalancer:'

get()

octavia.api.v2.controllers.member module

class MemberController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:member:'

delete(*id*)
Deletes a pool member.

get(*id*, *fields=None*)
Gets a single pool member's details.

get_all(*fields=None*)
Lists all pool members of a pool.

post(*member_*)
Creates a pool member on a pool.

put(*id*, *member_*)
Updates a pool member.

class MembersController(**args*, ***kwargs*)
Bases: `octavia.api.v2.controllers.member.MemberController`

put(*additive_only=False*, *members_=None*)
Updates all members.

octavia.api.v2.controllers.pool module

class PoolsController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:pool:'

delete(*id*)

Deletes a pool from a load balancer.

get(*id*, *fields=None*)

Gets a pool's details.

get_all(*project_id=None*, *fields=None*)

Lists all pools.

post(*pool_*)

Creates a pool on a load balancer or listener.

Note that this can optionally take a listener_id with which the pool should be associated as the listener's default_pool. If specified, the pool creation will fail if the listener specified already has a default_pool.

put(*id*, *pool_*)

Updates a pool on a load balancer.

octavia.api.v2.controllers.provider module

class AvailabilityZoneCapabilitiesController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:provider-availability-zone:'

get_all(*fields=None*)

class FlavorCapabilitiesController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:provider-flavor:'

get_all(*fields=None*)

class ProviderController(**args*, ***kwargs*)

Bases: `octavia.api.v2.controllers.base.BaseController`

RBAC_TYPE = 'os_load-balancer_api:provider:'

get_all(*fields=None*)

List enabled provider drivers and their descriptions.

octavia.api.v2.controllers.quotas module

class QuotasController(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:quota:'

delete(*project_id*)

Reset a project's quotas to the default values.

get(*project_id*)

Get a single project's quota details.

get_all(*project_id=None*)

List all non-default quotas.

put(*project_id*, *quotas*)

Update any or all quotas for a project.

class QuotasDefaultController(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

RBAC_TYPE = 'os_load-balancer_api:quota:'

get()

Get a project's default quota details.

Module contents

class BaseV2Controller(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

get()

healthmonitors = None

l7policies = None

listeners = None

loadbalancers = None

pools = None

quotas = None

class OctaviaV2Controller(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.base.BaseController*

amphorae = None

get()

class V2Controller(**args*, ***kwargs*)

Bases: *octavia.api.v2.controllers.BaseV2Controller*

lbaas = None

octavia.api.v2.types package

Submodules

octavia.api.v2.types.amphora module

class AmphoraResponse(***kw*)

Bases: *octavia.api.v2.types.amphora.BaseAmphoraType*

Defines which attributes are to be shown on any response.

cached_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

cert_busy

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

cert_expiration

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compute_flavor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compute_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

ha_ip

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

ha_port_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`image_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`lb_network_ip`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`loadbalancer_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

role

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vrrp_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vrrp_interface

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vrrp_ip

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vrrp_port_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vrrp_priority

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class` **AmphoraRootResponse**(***kw*)

Bases: *octavia.api.common.types.BaseType*

amphora

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AmphoraStatisticsResponse(***kw*)

Bases: `octavia.api.v2.types.amphora.BaseAmphoraType`

Defines which attributes are to show on stats response.

active_connections

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

bytes_in

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

bytes_out

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listener_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancer_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

request_errors

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

total_connections

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class **AmphoraeRootResponse**(***kw*)

Bases: *octavia.api.common.types.BaseType*

amphorae

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

amphorae_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class` **BaseAmphoraType**(***kw*)

Bases: *octavia.api.common.types.BaseType*

`class` **StatisticsRootResponse**(***kw*)

Bases: *octavia.api.common.types.BaseType*

amphora_stats

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.availability_zone_profile module

class AvailabilityZoneProfilePOST(***kw*)

Bases: *octavia.api.v2.types.availability_zone_profile.BaseAvailabilityZoneProfileType*

Defines mandatory and optional attributes of a POST request.

availability_zone_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneProfilePUT(***kw*)

Bases: *octavia.api.v2.types.availability_zone_profile.BaseAvailabilityZoneProfileType*

Defines the attributes of a PUT request.

availability_zone_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneProfileResponse(***kw*)

Bases: *octavia.api.v2.types.availability_zone_profile.BaseAvailabilityZoneProfileType*

Defines which attributes are to be shown on any response.

availability_zone_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneProfileRootPOST(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone_profile

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneProfileRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone_profile

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneProfileRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone_profile

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneProfilesRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone_profile_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

availability_zone_profiles

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class BaseAvailabilityZoneProfileType(***kw*)

Bases: *octavia.api.common.types.BaseType*

octavia.api.v2.types.availability_zones module

class AvailabilityZonePOST(***kw*)

Bases: *octavia.api.v2.types.availability_zones.BaseAvailabilityZoneType*

Defines mandatory and optional attributes of a POST request.

availability_zone_profile_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`enabled`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class AvailabilityZonePUT`(***kw*)

Bases: `octavia.api.v2.types.availability_zones.BaseAvailabilityZoneType`

Defines the attributes of a PUT request.

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`enabled`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class `AvailabilityZoneResponse`(***kw*)

Bases: *`octavia.api.v2.types.availability_zones.BaseAvailabilityZoneType`*

Defines which attributes are to be shown on any response.

`availability_zone_profile_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`enabled`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`classmethod from_data_model`(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **`data_model`** -- data model to convert from
- **`children`** -- convert child data models

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneRootPOST(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZoneRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class AvailabilityZonesRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

availability_zones

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

availability_zones_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class BaseAvailabilityZoneType(***kw*)

Bases: *octavia.api.common.types.BaseType*

octavia.api.v2.types.flavor_profile module

class BaseFlavorProfileType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class FlavorProfilePOST(***kw*)

Bases: *octavia.api.v2.types.flavor_profile.BaseFlavorProfileType*

Defines mandatory and optional attributes of a POST request.

flavor_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorProfilePUT(***kw*)

Bases: *octavia.api.v2.types.flavor_profile.BaseFlavorProfileType*

Defines the attributes of a PUT request.

flavor_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorProfileResponse(***kw*)

Bases: *octavia.api.v2.types.flavor_profile.BaseFlavorProfileType*

Defines which attributes are to be shown on any response.

flavor_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorProfileRootPOST(***kw*)

Bases: `octavia.api.common.types.BaseType`

flavorprofile

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorProfileRootPUT(***kw*)

Bases: `octavia.api.common.types.BaseType`

flavorprofile

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorProfileRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

flavorprofile

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorProfilesRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

flavorprofile_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

flavorprofiles

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.flavors module

class BaseFlavorType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class FlavorPOST(***kw*)

Bases: *octavia.api.v2.types.flavors.BaseFlavorType*

Defines mandatory and optional attributes of a POST request.

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

enabled

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`flavor_profile_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class FlavorPUT`(***kw*)

Bases: *`octavia.api.v2.types.flavors.BaseFlavorType`*

Defines the attributes of a PUT request.

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`enabled`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class FlavorResponse(`*`**kw`*`)`

Bases: *`octavia.api.v2.types.flavors.BaseFlavorType`*

Defines which attributes are to be shown on any response.

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`enabled`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`flavor_profile_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`classmethod from_data_model`(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **`data_model`** -- data model to convert from
- **`children`** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorRootPOST(***kw*)

Bases: *octavia.api.common.types.BaseType*

flavor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorRootPUT(***kw*)

Bases: `octavia.api.common.types.BaseType`

flavor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorRootResponse(***kw*)

Bases: `octavia.api.common.types.BaseType`

flavor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorsRootResponse(***kw*)

Bases: `octavia.api.common.types.BaseType`

flavors

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

flavors_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.health_monitor module

class BaseHealthMonitorType(***kw*)

Bases: `octavia.api.common.types.BaseType`

class HealthMonitorFullResponse(***kw*)

Bases: `octavia.api.v2.types.health_monitor.HealthMonitorResponse`

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

delay

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

domain_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

expected_codes

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_method

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_version

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries_down

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pools

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

timeout

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`type`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`updated_at`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`url_path`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorPOST(***kw*)

Bases: *octavia.api.v2.types.health_monitor.BaseHealthMonitorType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

delay

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

domain_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

expected_codes

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_method

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_version

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries_down

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

timeout

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

url_path

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorPUT(***kw*)

Bases: *octavia.api.v2.types.health_monitor.BaseHealthMonitorType*

Defines attributes that are acceptable of a PUT request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

delay

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

domain_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

expected_codes

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_method

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_version

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries_down

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`tags`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`timeout`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`url_path`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorResponse(***kw*)

Bases: *octavia.api.v2.types.health_monitor.BaseHealthMonitorType*

Defines which attributes are to be shown on any response.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

delay

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

domain_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

expected_codes

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

http_method

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

http_version

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`max_retries_down`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`operating_status`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pools

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

timeout

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

url_path

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorRootPOST(***kw*)

Bases: `octavia.api.common.types.BaseType`

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorSingleCreate(***kw*)

Bases: *octavia.api.v2.types.health_monitor.BaseHealthMonitorType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

delay

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

domain_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

expected_codes

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`http_method`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`http_version`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`max_retries`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

max_retries_down

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

url_path

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorStatusResponse(***kw*)

Bases: *octavia.api.v2.types.health_monitor.BaseHealthMonitorType*

Defines which attributes are to be shown on status response.

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class HealthMonitorsRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

healthmonitors

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitors_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.l7policy module

class BaseL7PolicyType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class L7PoliciesRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

l7policies

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7policies_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicyFullResponse(***kw*)

Bases: *octavia.api.v2.types.l7policy.L7PolicyResponse*

action

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`listener_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

position

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_http_code

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_prefix

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_url

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

rules

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicyPOST(***kw*)

Bases: *octavia.api.v2.types.l7policy.BaseL7PolicyType*

Defines mandatory and optional attributes of a POST request.

action

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listener_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

position

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_http_code

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_prefix

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_url

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

rules

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`tags`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`property tenant_id`

`class L7PolicyPUT``(**kw)`

Bases: *`octavia.api.v2.types.l7policy.BaseL7PolicyType`*

Defines attributes that are acceptable of a PUT request.

`action`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`admin_state_up`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

position

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_http_code

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_prefix

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_url

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicyResponse(***kw*)

Bases: `octavia.api.v2.types.l7policy.BaseL7PolicyType`

Defines which attributes are to be shown on any response.

action

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listener_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

position

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_http_code

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_prefix

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_url

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

rules

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicyRootPOST(***kw*)

Bases: `octavia.api.common.types.BaseType`

l7policy

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicyRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

l7policy

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicyRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

l7policy

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7PolicySingleCreate(***kw*)

Bases: *octavia.api.v2.types.l7policy.BaseL7PolicyType*

Defines mandatory and optional attributes of a POST request.

action

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

position

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_http_code

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_prefix

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

redirect_url

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

rules

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.l7rule module

class BaseL7Type(***kw*)

Bases: *octavia.api.common.types.BaseType*

class L7RuleFullResponse(***kw*)

Bases: *octavia.api.v2.types.l7rule.L7RuleResponse*

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compare_type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

invert

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

key

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

value

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RulePOST(***kw*)

Bases: *octavia.api.v2.types.l7rule.BaseL7Type*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compare_type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

invert

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

key

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

value

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RulePUT(***kw*)

Bases: *octavia.api.v2.types.l7rule.BaseL7Type*

Defines attributes that are acceptable of a PUT request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compare_type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

invert

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

key

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

value

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class` **L7RuleResponse**(***kw*)

Bases: *octavia.api.v2.types.l7rule.BaseL7Type*

Defines which attributes are to be shown on any response.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compare_type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

invert

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

key

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

value

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RuleRootPOST(***kw*)

Bases: *octavia.api.common.types.BaseType*

rule

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RuleRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

rule

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RuleRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

rule

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RuleSingleCreate(***kw*)

Bases: *octavia.api.v2.types.l7rule.BaseL7Type*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

compare_type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

invert

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

key

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

value

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class L7RulesRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

rules

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

rules_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.listener module

class BaseListenerType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class ListenerFullResponse(***kw*)

Bases: *octavia.api.v2.types.listener.ListenerResponse*

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

allowed_cidrs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_authentication

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

connection_limit

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

insert_headers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7policies

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

sni_container_refs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

timeout_client_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_connect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_tcp_inspect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerPOST(***kw*)

Bases: *octavia.api.v2.types.listener.BaseListenerType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

allowed_cidrs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_authentication

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

connection_limit

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

insert_headers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7policies

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancer_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`project_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`protocol`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

sni_container_refs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

timeout_client_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_connect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_tcp_inspect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerPUT(***kw*)

Bases: *octavia.api.v2.types.listener.BaseListenerType*

Defines attributes that are acceptable of a PUT request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

allowed_cidrs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_authentication

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

connection_limit

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

insert_headers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

sni_container_refs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_client_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_connect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_tcp_inspect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerResponse(***kw*)

Bases: `octavia.api.v2.types.listener.BaseListenerType`

Defines which attributes are to be shown on any response.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

allowed_cidrs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_authentication

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

connection_limit

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`created_at`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`default_pool_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`default_tls_container_ref`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`classmethod from_data_model`(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **`data_model`** -- data model to convert from
- **`children`** -- convert child data models

`id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`insert_headers`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`l7policies`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`loadbalancers`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

sni_container_refs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

timeout_client_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_connect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_tcp_inspect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerRootPOST(***kw*)

Bases: `octavia.api.common.types.BaseType`

listener

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

listener

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

listener

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerSingleCreate(***kw*)

Bases: *octavia.api.v2.types.listener.BaseListenerType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`allowed_cidrs`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`alpn_protocols`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`client_authentication`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

client_crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

connection_limit

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_pool_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

default_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

insert_headers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7policies

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

sni_container_refs

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_client_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_connect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_member_data

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

timeout_tcp_inspect

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenerStatisticsResponse(***kw*)

Bases: *octavia.api.v2.types.listener.BaseListenerType*

Defines which attributes are to show on stats response.

active_connections

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`bytes_in`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`bytes_out`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`classmethod from_data_model`(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **`data_model`** -- data model to convert from
- **`children`** -- convert child data models

`request_errors`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`total_connections`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class ListenerStatusResponse`(***kw*)

Bases: *`octavia.api.v2.types.listener.BaseListenerType`*

Defines which attributes are to be shown on status response.

`classmethod from_data_model`(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **`data_model`** -- data model to convert from
- **`children`** -- convert child data models

`id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pools

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ListenersRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

listeners

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listeners_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class StatisticsRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

stats

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.load_balancer module

class BaseLoadBalancerType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class LoadBalancerFullResponse(***kw*)

Bases: *octavia.api.v2.types.load_balancer.LoadBalancerResponse*

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

availability_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`created_at`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`flavor_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`listeners`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`operating_status`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`pools`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`project_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_network_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_port_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_qos_policy_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_subnet_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerFullRootResponse(***kw*)

Bases: *octavia.api.v2.types.load_balancer.LoadBalancerRootResponse*

loadbalancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerPOST(***kw*)

Bases: *octavia.api.v2.types.load_balancer.BaseLoadBalancerType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

availability_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`flavor_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`listeners`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pools

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

vip_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_network_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_port_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_qos_policy_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_subnet_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerPUT(***kw*)

Bases: `octavia.api.v2.types.load_balancer.BaseLoadBalancerType`

Defines attributes that are acceptable of a PUT request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_qos_policy_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerResponse(***kw*)

Bases: *octavia.api.v2.types.load_balancer.BaseLoadBalancerType*

Defines which attributes are to be shown on any response.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

availability_zone

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`flavor_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`classmethod from_data_model`(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **`data_model`** -- data model to convert from
- **`children`** -- convert child data models

`id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listeners

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pools

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provider

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_network_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_port_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_qos_policy_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

vip_subnet_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerRootPOST(***kw*)

Bases: *octavia.api.common.types.BaseType*

loadbalancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

loadbalancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

loadbalancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerStatisticsResponse(***kw*)

Bases: *octavia.api.v2.types.load_balancer.BaseLoadBalancerType*

Defines which attributes are to show on stats response.

active_connections

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

bytes_in

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

bytes_out

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

request_errors

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

total_connections

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancerStatusResponse(***kw*)

Bases: *octavia.api.v2.types.load_balancer.BaseLoadBalancerType*

Defines which attributes are to be shown on status response.

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listeners

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class LoadBalancersRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

loadbalancers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancers_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class StatisticsRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

stats

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class StatusResponse(***kw*)

Bases: `wsme.types.Base`

loadbalancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class StatusRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

statuses

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.member module

class BaseMemberType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class MemberFullResponse(***kw*)

Bases: *octavia.api.v2.types.member.MemberResponse*

address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

backup

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`subnet_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`tags`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`property tenant_id`

`updated_at`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

weight

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberPOST(***kw*)

Bases: *octavia.api.v2.types.member.BaseMemberType*

Defines mandatory and optional attributes of a POST request.

address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

backup

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

subnet_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

weight

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberPUT(***kw*)

Bases: *octavia.api.v2.types.member.BaseMemberType*

Defines attributes that are acceptable of a PUT request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

backup

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

weight

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberResponse(***kw*)

Bases: *octavia.api.v2.types.member.BaseMemberType*

Defines which attributes are to be shown on any response.

address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

backup

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

subnet_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

weight

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberRootPOST(***kw*)

Bases: `octavia.api.common.types.BaseType`

member

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

member

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

member

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberSingleCreate(***kw*)

Bases: *octavia.api.v2.types.member.BaseMemberType*

Defines mandatory and optional attributes of a POST request.

address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

backup

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

monitor_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`subnet_id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`tags`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`weight`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MemberStatusResponse(***kw*)

Bases: *octavia.api.v2.types.member.BaseMemberType*

Defines which attributes are to be shown on status response.

address

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol_port

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MembersRootPUT(***kw*)

Bases: *octavia.api.common.types.BaseType*

members

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class MembersRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

members

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

members_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.pool module

class BasePoolType(***kw*)

Bases: *octavia.api.common.types.BaseType*

class PoolFullResponse(***kw*)

Bases: *octavia.api.v2.types.pool.PoolResponse*

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitor_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

lb_algorithm

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listeners

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

members

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

session_persistence

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_enabled

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolPOST(***kw*)

Bases: *octavia.api.v2.types.pool.BasePoolType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

lb_algorithm

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listener_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancer_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

members

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

session_persistence

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_enabled

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolPUT(***kw*)

Bases: *octavia.api.v2.types.pool.BasePoolType*

Defines attributes that are acceptable of a PUT request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

lb_algorithm

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

session_persistence

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_enabled

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolResponse(***kw*)

Bases: *octavia.api.v2.types.pool.BasePoolType*

Defines which attributes are to be shown on any response.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

ca_tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

created_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

crl_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

healthmonitor_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

lb_algorithm

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listeners

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

members

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

protocol

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

session_persistence

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_enabled

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

updated_at

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolRootPOST(*kw*)**

Bases: *octavia.api.common.types.BaseType*

pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolRootPut(*kw*)**

Bases: *octavia.api.common.types.BaseType*

pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolSingleCreate(***kw*)

Bases: *octavia.api.v2.types.pool.BasePoolType*

Defines mandatory and optional attributes of a POST request.

admin_state_up

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

alpn_protocols

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`ca_tls_container_ref`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`crl_container_ref`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`description`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

lb_algorithm

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

members

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`protocol`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`session_persistence`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tags

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_ciphers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_container_ref

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_enabled

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

tls_versions

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolStatusResponse(***kw*)

Bases: `octavia.api.v2.types.pool.BasePoolType`

Defines which attributes are to be shown on status response.

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

health_monitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`id`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`members`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

operating_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

provisioning_status

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class PoolsRootResponse(***kw*)

Bases: `octavia.api.common.types.BaseType`

pools

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pools_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class SessionPersistencePOST(***kw*)

Bases: *octavia.api.common.types.BaseType*

Defines mandatory and optional attributes of a POST request.

cookie_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

persistence_granularity

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`persistence_timeout`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`type`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`class SessionPersistencePUT`(***kw*)

Bases: *`octavia.api.common.types.BaseType`*

Defines attributes that are acceptable of a PUT request.

`cookie_name`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

persistence_granularity

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

persistence_timeout

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

type

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class SessionPersistenceResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

Defines which attributes are to be shown on any response.

cookie_name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

persistence_granularity

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

persistence_timeout

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`type`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.provider module

`class AvailabilityZoneCapabilitiesResponse`(***kw*)

Bases: *`octavia.api.common.types.BaseType`*

`availability_zone_capabilities`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class FlavorCapabilitiesResponse(***kw*)

Bases: `octavia.api.common.types.BaseType`

flavor_capabilities

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ProviderResponse(***kw*)

Bases: `octavia.api.common.types.BaseType`

description

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

name

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class ProvidersRootResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

providers

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

octavia.api.v2.types.quotas module

class QuotaAllBase(***kw*)

Bases: *octavia.api.common.types.BaseType*

Wrapper object for get all quotas responses.

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

health_monitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7policy

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7rule

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

listener

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

load_balancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

loadbalancer

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

member

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

pool

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

project_id

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

property tenant_id

class QuotaAllResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

quotas

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

quotas_links

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class QuotaBase(***kw*)

Bases: *octavia.api.common.types.BaseType*

Individual quota definitions.

health_monitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

healthmonitor

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7policy

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

l7rule

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`listener`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`load_balancer`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`loadbalancer`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
```

(continues on next page)

(continued from previous page)

```
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`member`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`pool`

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

`to_dict`(*render_unsets=False*)

Converts Octavia WSME type to dictionary.

Parameters **`render_unsets`** -- If True, will convert items that are WSME Unset types to None. If False, does not add the item

`class QuotaPUT`(***kw*)

Bases: *`octavia.api.common.types.BaseType`*

Overall object for quota PUT request.

quota

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

class QuotaResponse(***kw*)

Bases: *octavia.api.common.types.BaseType*

Wrapper object for quotas responses.

classmethod from_data_model(*data_model*, *children=False*)

Converts data_model to Octavia WSME type.

Parameters

- **data_model** -- data model to convert from
- **children** -- convert child data models

quota

Complex type attribute definition.

Example:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = int
    mandatoryvalue = wsattr(int, mandatory=True)
    named_value = wsattr(int, name='named.value')
```

After inspection, the non-wsattr attributes will be replaced, and the above class will be equivalent to:

```
class MyComplexType(wsme.types.Base):
    optionalvalue = wsattr(int)
    mandatoryvalue = wsattr(int, mandatory=True)
```

Module contents

Module contents

Submodules

octavia.api.app module

get_pecan_config()
Returns the pecan config.

setup_app(*pecan_config=None*, *debug=False*, *argv=None*)
Creates and returns a pecan wsgi app.

octavia.api.config module

octavia.api.root_controller module

class RootController
Bases: `object`

The controller with which the pecan wsgi app should be created.

healthcheck()

index()

Module contents

octavia.certificates package

Subpackages

octavia.certificates.common package

Subpackages

octavia.certificates.common.auth package

Submodules

octavia.certificates.common.auth.barbican_acl module

Barbican ACL auth class for Barbican certificate handling

class BarbicanACLAuth
Bases: *`octavia.certificates.common.barbican.BarbicanAuth`*

classmethod ensure_secret_access(*context*, *ref*)

Do whatever steps are necessary to ensure future access to a secret.

Parameters

- **context** -- pecan context object
- **ref** -- Reference to a Barbican object

classmethod get_barbican_client(*project_id=None*)

Creates a Barbican client object.

Parameters **project_id** -- Project ID that the request will be used for

Returns a Barbican Client object

Raises **Exception** -- if the client cannot be created

classmethod get_barbican_client_user_auth(*context*)

classmethod revoke_secret_access(*context*, *ref*)

Revoke access of Octavia keystone user to a secret.

Parameters

- **context** -- pecan context object
- **ref** -- Reference to a Barbican object

Module contents

Submodules

octavia.certificates.common.barbican module

Common classes for Barbican certificate handling

class BarbicanAuth

Bases: `object`

abstract ensure_secret_access(*context*, *ref*)

Do whatever steps are necessary to ensure future access to a secret.

Parameters

- **context** -- pecan context object
- **ref** -- Reference to a Barbican object

abstract get_barbican_client(*project_id*)

Creates a Barbican client object.

Parameters **project_id** -- Project ID that the request will be used for

Returns a Barbican Client object

Raises **Exception** -- if the client cannot be created

abstract revoke_secret_access(*context*, *ref*)

Revoke access of Octavia keystone user to a secret.

Parameters

- **context** -- pecan context object
- **ref** -- Reference to a Barbican object

`class BarbicanCert(cert_container)`

Bases: `octavia.certificates.common.cert.Cert`

Representation of a Cert based on the Barbican CertificateContainer.

`get_certificate()`

Returns the certificate.

`get_intermediates()`

Returns the intermediate certificates as a list.

`get_private_key()`

Returns the private key for the certificate.

`get_private_key_passphrase()`

Returns the passphrase for the private key.

octavia.certificates.common.cert module

`class Cert`

Bases: `object`

Base class to represent all certificates.

`abstract get_certificate()`

Returns the certificate.

`abstract get_intermediates()`

Returns the intermediate certificates as a list.

`abstract get_private_key()`

Returns the private key for the certificate.

`abstract get_private_key_passphrase()`

Returns the passphrase for the private key.

octavia.certificates.common.local module

Common classes for local filesystem certificate handling

`class LocalCert(certificate, private_key, intermediates=None, private_key_passphrase=None)`

Bases: `octavia.certificates.common.cert.Cert`

Representation of a Cert for local storage.

`get_certificate()`

Returns the certificate.

`get_intermediates()`

Returns the intermediate certificates as a list.

get_private_key()
Returns the private key for the certificate.

get_private_key_passphrase()
Returns the passphrase for the private key.

octavia.certificates.common.pkcs12 module

Common classes for pkcs12 based certificate handling

class PKCS12Cert(*certbag*)
Bases: `octavia.certificates.common.cert.Cert`

Representation of a Cert for local storage.

get_certificate()
Returns the certificate.

get_intermediates()
Returns the intermediate certificates as a list.

get_private_key()
Returns the private key for the certificate.

get_private_key_passphrase()
Returns the passphrase for the private key.

Module contents

octavia.certificates.generator package

Submodules

octavia.certificates.generator.cert_gen module

Certificate Generator API

class CertGenerator
Bases: `object`

Base Cert Generator Interface

A Certificate Generator is responsible for generating private keys, generating CSRs, and signing TLS certificates.

abstract generate_cert_key_pair(*cn*, *validity*, *bit_length*, *passphrase*)
Generates a private key and certificate pair

Parameters

- **cn** -- Common name to use for the Certificate
- **validity** -- Validity period for the Certificate
- **bit_length** -- Private key bit length

- **passphrase** -- Passphrase to use for encrypting the private key

Returns octavia.certificates.common.Cert representation of the certificate data

Raises **Exception** -- If generation fails

abstract sign_cert(*csr*, *validity*)

Generates a signed certificate from the provided CSR

This call is designed to block until a signed certificate can be returned.

Parameters

- **csr** -- A Certificate Signing Request
- **validity** -- Valid for <validity> seconds from the current time

Returns PEM Encoded Signed certificate

Raises **Exception** -- If certificate signing fails

octavia.certificates.generator.local module

class LocalCertGenerator

Bases: *octavia.certificates.generator.cert_gen.CertGenerator*

Cert Generator Interface that signs certs locally.

classmethod generate_cert_key_pair(*cn*, *validity*, *bit_length=2048*, *passphrase=None*, ***kwargs*)

Generates a private key and certificate pair

Parameters

- **cn** -- Common name to use for the Certificate
- **validity** -- Validity period for the Certificate
- **bit_length** -- Private key bit length
- **passphrase** -- Passphrase to use for encrypting the private key

Returns octavia.certificates.common.Cert representation of the certificate data

Raises **Exception** -- If generation fails

classmethod sign_cert(*csr*, *validity*, *ca_cert=None*, *ca_key=None*, *ca_key_pass=None*, *ca_digest=None*)

Signs a certificate using our private CA based on the specified CSR

The signed certificate will be valid from now until <validity> seconds from now.

Parameters

- **csr** -- A Certificate Signing Request
- **validity** -- Valid for <validity> seconds from the current time
- **ca_cert** -- Signing Certificate (default: config)
- **ca_key** -- Signing Certificate Key (default: config)
- **ca_key_pass** -- Signing Certificate Key Pass (default: config)

- **`ca_digest`** -- Digest method to use for signing (default: config)

Returns Signed certificate

Raises **`Exception`** -- if certificate signing fails

Module contents

octavia.certificates.manager package

Submodules

octavia.certificates.manager.barbican module

Cert manager implementation for Barbican using a single PKCS12 secret

`class BarbicanCertManager`

Bases: *`octavia.certificates.manager.cert_mgr.CertManager`*

Certificate Manager that wraps the Barbican client API.

`delete_cert`(*context*, *cert_ref*, *resource_ref*, *service_name=None*)

Deregister as a consumer for the specified cert.

Parameters

- **`context`** -- Oslo context of the request
- **`cert_ref`** -- the UUID of the cert to retrieve
- **`resource_ref`** -- Full HATEOAS reference to the consuming resource
- **`service_name`** -- Friendly name for the consuming service

Raises **`Exception`** -- if deregistration fails

`get_cert`(*context*, *cert_ref*, *resource_ref=None*, *check_only=False*, *service_name=None*)

Retrieves the specified cert and registers as a consumer.

Parameters

- **`context`** -- Oslo context of the request
- **`cert_ref`** -- the UUID of the cert to retrieve
- **`resource_ref`** -- Full HATEOAS reference to the consuming resource
- **`check_only`** -- Read Certificate data without registering
- **`service_name`** -- Friendly name for the consuming service

Returns octavia.certificates.common.Cert representation of the certificate data

Raises **`Exception`** -- if certificate retrieval fails

`get_secret`(*context*, *secret_ref*)

Retrieves a secret payload by reference.

Parameters

- **`context`** -- Oslo context of the request

- **`secret_ref`** -- The secret reference ID

Returns The secret payload

Raises `CertificateStorageException` -- if retrieval fails

`set_acls`(*context*, *cert_ref*)

Adds ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the addition of ACLs fails for any reason, a CertificateStorageException should be raised.

`store_cert`(*context*, *certificate*, *private_key*, *intermediates=None*, *private_key_passphrase=None*, *expiration=None*, *name='PKCS12 Certificate Bundle'*)

Stores a certificate in the certificate manager.

Parameters

- **`context`** -- Oslo context of the request
- **`certificate`** -- PEM encoded TLS certificate
- **`private_key`** -- private key for the supplied certificate
- **`intermediates`** -- ordered and concatenated intermediate certs
- **`private_key_passphrase`** -- optional passphrase for the supplied key
- **`expiration`** -- the expiration time of the cert in ISO 8601 format
- **`name`** -- a friendly name for the cert

Returns the container_ref of the stored cert

Raises `Exception` -- if certificate storage fails

`unset_acls`(*context*, *cert_ref*)

Remove ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the removal of ACLs fails for any reason, a CertificateStorageException should be raised.

octavia.certificates.manager.barbican_legacy module

Legacy cert manager implementation for Barbican (container+secrets)

`class BarbicanCertManager`(*auth=None*)

Bases: `octavia.certificates.manager.cert_mgr.CertManager`

Certificate Manager that wraps the Barbican client API.

`delete_cert`(*context*, *cert_ref*, *resource_ref*, *service_name=None*)

Deregister as a consumer for the specified cert.

Parameters

- **`context`** -- Oslo context of the request
- **`cert_ref`** -- the UUID of the cert to retrieve
- **`resource_ref`** -- Full HATEOAS reference to the consuming resource

- **service_name** -- Friendly name for the consuming service

Raises `Exception` -- if deregistration fails

get_cert(*context*, *cert_ref*, *resource_ref=None*, *check_only=False*, *service_name=None*)

Retrieves the specified cert and registers as a consumer.

Parameters

- **context** -- Oslo context of the request
- **cert_ref** -- the UUID of the cert to retrieve
- **resource_ref** -- Full HATEOAS reference to the consuming resource
- **check_only** -- Read Certificate data without registering
- **service_name** -- Friendly name for the consuming service

Returns octavia.certificates.common.Cert representation of the certificate data

Raises `Exception` -- if certificate retrieval fails

get_secret(*context*, *secret_ref*)

Retrieves a secret payload by reference.

If the specified secret does not exist, a CertificateStorageException should be raised.

set_acls(*context*, *cert_ref*)

Adds ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the addition of ACLs fails for any reason, a CertificateStorageException should be raised.

store_cert(*context*, *certificate*, *private_key*, *intermediates=None*, *private_key_passphrase=None*, *expiration=None*, *name=None*)

Stores a certificate in the certificate manager.

Parameters

- **context** -- Oslo context of the request
- **certificate** -- PEM encoded TLS certificate
- **private_key** -- private key for the supplied certificate
- **intermediates** -- ordered and concatenated intermediate certs
- **private_key_passphrase** -- optional passphrase for the supplied key
- **expiration** -- the expiration time of the cert in ISO 8601 format
- **name** -- a friendly name for the cert

Returns the container_ref of the stored cert

Raises `Exception` -- if certificate storage fails

unset_acls(*context*, *cert_ref*)

Remove ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the removal of ACLs fails for any reason, a CertificateStorageException should be raised.

octavia.certificates.manager.castellan_mgr module

Cert manager implementation for Castellan

class CastellanCertManager

Bases: `octavia.certificates.manager.cert_mgr.CertManager`

Certificate Manager for the Castellan library.

delete_cert(*context*, *cert_ref*, *resource_ref*, *service_name=None*)

Deletes the specified cert.

If the specified cert does not exist, a CertificateStorageException should be raised.

get_cert(*context*, *cert_ref*, *resource_ref=None*, *check_only=False*, *service_name=None*)

Retrieves the specified cert.

If check_only is True, don't perform any sort of registration. If the specified cert does not exist, a CertificateStorageException should be raised.

get_secret(*context*, *secret_ref*)

Retrieves a secret payload by reference.

If the specified secret does not exist, a CertificateStorageException should be raised.

set_acls(*context*, *cert_ref*)

Adds ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the addition of ACLs fails for any reason, a CertificateStorageException should be raised.

store_cert(*context*, *certificate*, *private_key*, *intermediates=None*, *private_key_passphrase=None*, *expiration=None*, *name='PKCS12 Certificate Bundle'*)

Stores (i.e., registers) a cert with the cert manager.

This method stores the specified cert and returns its UUID that identifies it within the cert manager. If storage of the certificate data fails, a CertificateStorageException should be raised.

unset_acls(*context*, *cert_ref*)

Remove ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the removal of ACLs fails for any reason, a CertificateStorageException should be raised.

octavia.certificates.manager.cert_mgr module

Certificate manager API

class CertManager

Bases: `object`

Base Cert Manager Interface

A Cert Manager is responsible for managing certificates for TLS.

abstract delete_cert(*context*, *cert_ref*, *resource_ref*, *service_name=None*)

Deletes the specified cert.

If the specified cert does not exist, a CertificateStorageException should be raised.

abstract get_cert(*context*, *cert_ref*, *resource_ref=None*, *check_only=False*, *service_name=None*)

Retrieves the specified cert.

If check_only is True, don't perform any sort of registration. If the specified cert does not exist, a CertificateStorageException should be raised.

abstract get_secret(*context*, *secret_ref*)

Retrieves a secret payload by reference.

If the specified secret does not exist, a CertificateStorageException should be raised.

abstract set_acls(*context*, *cert_ref*)

Adds ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the addition of ACLs fails for any reason, a CertificateStorageException should be raised.

abstract store_cert(*context*, *certificate*, *private_key*, *intermediates=None*, *private_key_passphrase=None*, *expiration=None*, *name=None*)

Stores (i.e., registers) a cert with the cert manager.

This method stores the specified cert and returns its UUID that identifies it within the cert manager. If storage of the certificate data fails, a CertificateStorageException should be raised.

abstract unset_acls(*context*, *cert_ref*)

Remove ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the removal of ACLs fails for any reason, a CertificateStorageException should be raised.

octavia.certificates.manager.local module

class LocalCertManager

Bases: `octavia.certificates.manager.cert_mgr.CertManager`

Cert Manager Interface that stores data locally.

static delete_cert(*context*, *cert_ref*, ***kwargs*)

Deletes the specified cert.

Parameters

- **context** -- Ignored in this implementation
- **cert_ref** -- the UUID of the cert to delete

Raises `CertificateStorageException` -- if certificate deletion fails

static get_cert(*context*, *cert_ref*, ***kwargs*)

Retrieves the specified cert.

Parameters

- **context** -- Ignored in this implementation
- **cert_ref** -- the UUID of the cert to retrieve

Returns octavia.certificates.common.Cert representation of the certificate data

Raises *CertificateStorageException* -- if certificate retrieval fails

static get_secret(*context*, *secret_ref*)

Retrieves a secret payload by reference.

Parameters

- **context** -- Ignored in this implementation
- **secret_ref** -- The secret reference ID

Returns The secret payload

Raises *CertificateStorageException* -- if secret retrieval fails

set_acls(*context*, *cert_ref*)

Adds ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the addition of ACLs fails for any reason, a CertificateStorageException should be raised.

static store_cert(*context*, *certificate*, *private_key*, *intermediates=None*, *private_key_passphrase=None*, ***kwargs*)

Stores (i.e., registers) a cert with the cert manager.

This method stores the specified cert to the filesystem and returns a UUID that can be used to retrieve it.

Parameters

- **context** -- Ignored in this implementation
- **certificate** -- PEM encoded TLS certificate
- **private_key** -- private key for the supplied certificate
- **intermediates** -- ordered and concatenated intermediate certs
- **private_key_passphrase** -- optional passphrase for the supplied key

Returns the UUID of the stored cert

Raises *CertificateStorageException* -- if certificate storage fails

unset_acls(*context*, *cert_ref*)

Remove ACLs so Octavia can access the cert objects.

If the specified cert does not exist or the removal of ACLs fails for any reason, a CertificateStorageException should be raised.

Module contents

Module contents

octavia.cmd package

Submodules

octavia.cmd.agent module

class AmphoraAgent(*app*, *options=None*)

Bases: `gunicorn.app.base.BaseApplication`

load()

load_config()

This method is used to load the configuration from one or several input(s). Custom Command line, configuration file. You have to override this method in your class.

main()

octavia.cmd.api module

main()

octavia.cmd.driver_agent module

main()

octavia.cmd.haproxy_vrrp_check module

get_status(*sock_address*)

Query haproxy stat socket

Only VRRP fail over if the stats socket is not responding.

Parameters **sock_address** -- unix socket file

Returns 0 if haproxy responded

health_check(*sock_addresses*)

Invoke queries for all defined listeners

Parameters **sock_addresses** --

Returns

main()

octavia.cmd.health_checker module

`crc32c`(*data*)

`main`()

`sctp_health_check`(*ip_address*, *port*, *timeout=2*)

octavia.cmd.health_manager module

`hm_health_check`(*exit_event*)

`hm_listener`(*exit_event*)

`main`()

octavia.cmd.house_keeping module

`cert_rotation`()

Perform certificate rotation.

`db_cleanup`()

Perform db cleanup for old resources.

`main`()

octavia.cmd.interface module

`exception` **`InterfaceException`**(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message` `= 'Could not configure interface:  %(msg)s'`

`interface_cmd`(*interface_name*, *action*)

`interfaces_find`(*interface_controller*, *name*)

`interfaces_update`(*interfaces*, *action_fn*, *action_str*)

`main`()

octavia.cmd.octavia_worker module

`main`()

octavia.cmd.status module

`class` **`Checks`**

Bases: `oslo_upgradecheck.upgradecheck.UpgradeCommands`

Contains upgrade checks

Various upgrade checks should be added as separate methods in this class and added to _upgrade_checks tuple.

`main``()`

Module contents

octavia.common package

Subpackages

octavia.common.jinja package

Subpackages

octavia.common.jinja.haproxy package

Subpackages

octavia.common.jinja.haproxy.combined_listeners package

Submodules

octavia.common.jinja.haproxy.combined_listeners.jinja_cfg module

`class` **`JinjaTemplater`**(*base_amp_path=None*, *base_crt_dir=None*, *haproxy_template=None*, *log_http=None*, *log_server=None*, *connection_logging=True*)

Bases: `object`

`build_config`(*host_amphora*, *listeners*, *tls_certs*, *haproxy_versions*, *socket_path=None*)

Convert a logical configuration to the HAProxy version

Parameters

- **`host_amphora`** -- The Amphora this configuration is hosted on
- **`listener`** -- The listener configuration
- **`socket_path`** -- The socket path for Haproxy process

Returns Rendered configuration

`render_loadbalancer_obj`(*host_amphora*, *listeners*, *tls_certs=None*, *socket_path=None*, *feature_compatibility=None*)

Renders a templated configuration from a load balancer object

Parameters

- **`host_amphora`** -- The Amphora this configuration is hosted on
- **`listener`** -- The listener configuration
- **`tls_certs`** -- Dict of the TLS certificates for the listener
- **`socket_path`** -- The socket path for Haproxy process

Returns Rendered configuration

Module contents

octavia.common.jinja.haproxy.split_listeners package

Submodules

octavia.common.jinja.haproxy.split_listeners.jinja_cfg module

`class JinjaTemplater`(*base_amp_path=None*, *base_crt_dir=None*, *haproxy_template=None*, *log_http=None*, *log_server=None*, *connection_logging=True*)

Bases: `object`

`build_config`(*host_amphora*, *listener*, *haproxy_versions*, *socket_path=None*, *client_ca_filename=None*, *client_crl=None*, *pool_tls_certs=None*)

Convert a logical configuration to the HAProxy version

Parameters

- **`host_amphora`** -- The Amphora this configuration is hosted on
- **`listener`** -- The listener configuration
- **`socket_path`** -- The socket path for Haproxy process

Returns Rendered configuration

`render_loadbalancer_obj`(*host_amphora*, *listener*, *socket_path=None*, *feature_compatibility=None*, *client_ca_filename=None*, *client_crl=None*, *pool_tls_certs=None*)

Renders a templated configuration from a load balancer object

Parameters

- **`host_amphora`** -- The Amphora this configuration is hosted on
- **`listener`** -- The listener configuration
- **`client_ca_filename`** -- The CA certificate for client authorization
- **`socket_path`** -- The socket path for Haproxy process

Returns Rendered configuration

Module contents

Module contents

octavia.common.jinja.logging package

Submodules

octavia.common.jinja.logging.logging_jinja_cfg module

`class` **`LoggingJinjaTemplater`**`(`*logging_templates=None*`)`

Bases: `object`

`build_logging_config``()`

Module contents

octavia.common.jinja.lvs package

Submodules

octavia.common.jinja.lvs.jinja_cfg module

`class` **`LvsJinjaTemplater`**`(`*base_amp_path=None*, *keepalivedlvs_template=None*`)`

Bases: `object`

`build_config``(`*listener*, ***kwargs*`)`

Convert a logical configuration to the Keepalived LVS version

Parameters **`listener`** -- The listener configuration

Returns Rendered configuration

`render_loadbalancer_obj``(`*listener*, ***kwargs*`)`

Renders a templated configuration from a load balancer object

Parameters

- **`host_amphora`** -- The Amphora this configuration is hosted on
- **`listener`** -- The listener configuration

Returns Rendered configuration

Module contents

Submodules

octavia.common.jinja.user_data_jinja_cfg module

class UserDataJinjaCfg

Bases: `object`

build_user_data_config(*user_data*)

Module contents

octavia.common.tls_utils package

Submodules

octavia.common.tls_utils.cert_parser module

build_pem(*tls_container*)

Concatenate TLS container fields to create a PEM

encoded certificate file

Parameters **tls_container** -- Object container TLS certificates

Returns Pem encoded certificate file

get_cert_expiration(*certificate_pem*)

Extract the expiration date from the Pem encoded X509 certificate

Parameters **certificate_pem** -- Certificate in PEM format

Returns Expiration date of certificate_pem

get_host_names(*certificate*)

Extract the host names from the Pem encoded X509 certificate

Parameters **certificate** -- A PEM encoded certificate

Returns A dictionary containing the following keys: ['cn', 'dns_names'] where 'cn' is the CN from the SubjectName of the certificate, and 'dns_names' is a list of dNSNames (possibly empty) from the SubjectAltNames of the certificate.

get_intermediates_pems(*intermediates=None*)

Split the input string into individual x509 text blocks

Parameters **intermediates** -- PEM or PKCS7 encoded intermediate certificates

Returns A list of strings where each string represents an X509 pem block surrounded by BEGIN CERTIFICATE, END CERTIFICATE block tags

get_primary_cn(*tls_cert*)

Returns primary CN for Certificate.

load_certificates_data(*cert_mngr*, *obj*, *context=None*)

Load TLS certificate data from the listener/pool.

return TLS_CERT and SNI_CERTS

prepare_private_key(*private_key*, *passphrase=None*)

Prepares an unencrypted PEM-encoded private key for printing

Parameters **private_key** -- The private key in PEM format (encrypted or not)

Returns The unencrypted private key in PEM format

validate_cert(*certificate*, *private_key=None*, *private_key_passphrase=None*, *intermediates=None*)

Validate that the certificate is a valid PEM encoded X509 object

Optionally verify that the private key matches the certificate. Optionally verify that the intermediates are valid X509 objects.

Parameters

- **certificate** -- A PEM encoded certificate
- **private_key** -- The private key for the certificate
- **private_key_passphrase** -- Passphrase for accessing the private key
- **intermediates** -- PEM or PKCS7 encoded intermediate certificates

Returns boolean

Module contents

Submodules

octavia.common.base_taskflow module

class **BaseTaskFlowEngine**

Bases: `object`

This is the task flow engine

Use this engine to start/load flows in the code

taskflow_load(*flow*, ***kwargs*)

class **DynamicLoggingConductor**(*name*, *jobboard*, *persistence=None*, *engine=None*, *engine_options=None*, *wait_timeout=None*, *log=None*, *max_simultaneous_jobs=1*)

Bases: `taskflow.conductors.backends.impl_blocking.BlockingConductor`

class **ExtendExpiryListener**(*engine*, *job*)

Bases: `taskflow.listeners.base.Listener`

class **FilteredJob**(*board*, *name*, *uuid=None*, *details=None*, *backend=None*, *book=None*, *book_data=None*)

Bases: `taskflow.jobs.base.Job`

class **JobDetailsFilter**(*name=''*)

Bases: `logging.Filter`

filter(*record*)

Determine if the specified record is to be logged.

Returns True if the record should be logged, or False otherwise. If deemed appropriate, the record may be modified in-place.

class RedisDynamicLoggingConductor(*name*, *jobboard*, *persistence=None*, *engine=None*, *engine_options=None*, *wait_timeout=None*, *log=None*, *max_simultaneous_jobs=1*)

Bases: `octavia.common.base_taskflow.DynamicLoggingConductor`

class TaskFlowServiceController(*driver*)

Bases: `object`

run_conductor(*name*)

run_poster(*flow_factory*, **args*, ***kwargs*)

retryMaskFilter(*record*)

octavia.common.clients module

class CinderAuth

Bases: `object`

cinder_client = None

classmethod get_cinder_client(*region*, *service_name=None*, *endpoint=None*, *endpoint_type='publicURL'*, *insecure=False*, *cacert=None*)

Create cinder client object.

Parameters

- **region** -- The region of the service
- **service_name** -- The name of the cinder service in the catalog
- **endpoint** -- The endpoint of the service
- **endpoint_type** -- The endpoint type of the service
- **insecure** -- Turn off certificate validation
- **cacert** -- CA Cert file path

Returns a Cinder Client object

Raises `Exception` -- if the client cannot be created

class GlanceAuth

Bases: `object`

classmethod get_glance_client(*region*, *service_name=None*, *endpoint=None*, *endpoint_type='publicURL'*, *insecure=False*, *cacert=None*)

Create glance client object.

Parameters

- **region** -- The region of the service

- `service_name` -- The name of the glance service in the catalog
- `endpoint` -- The endpoint of the service
- `endpoint_type` -- The endpoint_type of the service
- `insecure` -- Turn off certificate validation
- `cacert` -- CA Cert file path

Returns a Glance Client object.

Raises `Exception` -- if the client cannot be created

`glance_client = <glanceclient.v2.client.Client object>`

`class NeutronAuth`

Bases: `object`

`classmethod get_neutron_client`(*region*, *service_name=None*, *endpoint=None*, *endpoint_type='publicURL'*, *insecure=False*, *ca_cert=None*)

Create neutron client object.

Parameters

- `region` -- The region of the service
- `service_name` -- The name of the neutron service in the catalog
- `endpoint` -- The endpoint of the service
- `endpoint_type` -- The endpoint_type of the service
- `insecure` -- Turn off certificate validation
- `ca_cert` -- CA Cert file path

Returns a Neutron Client object.

Raises `Exception` -- if the client cannot be created

`classmethod get_user_neutron_client`(*context*)

Get neutron client for request user.

It's possible that the token in the context is a trust scoped which can't be used to initialize a keystone session.

We directly use the token and endpoint_url to initialize neutron client.

`neutron_client = None`

`class NovaAuth`

Bases: `object`

`classmethod get_nova_client`(*region*, *service_name=None*, *endpoint=None*, *endpoint_type='publicURL'*, *insecure=False*, *cacert=None*)

Create nova client object.

Parameters

- `region` -- The region of the service
- `service_name` -- The name of the nova service in the catalog

- **endpoint** -- The endpoint of the service
- **endpoint_type** -- The type of the endpoint
- **insecure** -- Turn off certificate validation
- **cacert** -- CA Cert file path

Returns a Nova Client object.

Raises `Exception` -- if the client cannot be created

nova_client = <novaclient.v2.client.Client object>

octavia.common.config module

Routines for configuring Octavia

init(*args*, ***kwargs*)

register_cli_opts()

set_lib_defaults()

Update default value for configuration options from other namespace.

Example, oslo lib config options. This is needed for config generator tool to pick these default value changes. https://docs.openstack.org/oslo.config/latest/cli/ generator.html#modifying-defaults-from-other-namespaces

setup_logging(*conf*)

Sets up the logging options for a log with supplied name.

Parameters **conf** -- a cfg.ConfOpts object

setup_remote_debugger()

Required setup for remote debugging.

octavia.common.constants module

octavia.common.context module

class Context(*user_id=None*, *project_id=None*, ***kwargs*)

Bases: `oslo_context.context.RequestContext`

property project_id

property session

octavia.common.data_models module

class Amphora(*id=None*, *load_balancer_id=None*, *compute_id=None*, *status=None*, *lb_network_ip=None*, *vrrp_ip=None*, *ha_ip=None*, *vrrp_port_id=None*, *ha_port_id=None*, *load_balancer=None*, *role=None*, *cert_expiration=None*, *cert_busy=False*, *vrrp_interface=None*, *vrrp_id=None*, *vrrp_priority=None*, *cached_zone=None*, *created_at=None*, *updated_at=None*, *image_id=None*, *compute_flavor=None*)

Bases: `octavia.common.data_models.`

delete()

class AmphoraHealth(*amphora_id=None*, *last_update=None*, *busy=False*)

Bases: `octavia.common.data_models.`

class AvailabilityZone(*name=None*, *description=None*, *enabled=None*, *availability_zone_profile_id=None*)

Bases: `octavia.common.data_models.`

class AvailabilityZoneProfile(*id=None*, *name=None*, *provider_name=None*, *availability_zone_data=None*)

Bases: `octavia.common.data_models.`

class BaseDataModel

Bases: `object`

classmethod from_dict(*dict*)

to_dict(*calling_classes=None*, *recurse=False*, ***kwargs*)

Converts a data model to a dictionary.

update(*update_dict*)

Generic update method which works for simple,

non-relational attributes.

class Flavor(*id=None*, *name=None*, *description=None*, *enabled=None*, *flavor_profile_id=None*)

Bases: `octavia.common.data_models.`

class FlavorProfile(*id=None*, *name=None*, *provider_name=None*, *flavor_data=None*)

Bases: `octavia.common.data_models.`

class HealthMonitor(*id=None*, *project_id=None*, *pool_id=None*, *type=None*, *delay=None*, *timeout=None*, *fall_threshold=None*, *rise_threshold=None*, *http_method=None*, *url_path=None*, *expected_codes=None*, *enabled=None*, *pool=None*, *name=None*, *provisioning_status=None*, *operating_status=None*, *created_at=None*, *updated_at=None*, *tags=None*, *http_version=None*, *domain_name=None*)

Bases: `octavia.common.data_models.`

delete()

class L7Policy(*id=None*, *name=None*, *description=None*, *listener_id=None*, *action=None*, *redirect_pool_id=None*, *redirect_url=None*, *position=None*, *listener=None*, *redirect_pool=None*, *enabled=None*, *l7rules=None*, *provisioning_status=None*, *operating_status=None*, *project_id=None*, *created_at=None*, *updated_at=None*, *redirect_prefix=None*, *tags=None*, *redirect_http_code=None*)

Bases: `octavia.common.data_models.`

delete()

update(*update_dict*)

Generic update method which works for simple,

non-relational attributes.

class **L7Rule**(*id=None*, *l7policy_id=None*, *type=None*, *enabled=None*, *compare_type=None*, *key=None*, *value=None*, *l7policy=None*, *invert=False*, *provisioning_status=None*, *operating_status=None*, *project_id=None*, *created_at=None*, *updated_at=None*, *tags=None*)

Bases: `octavia.common.data_models.`

delete()

class **Listener**(*id=None*, *project_id=None*, *name=None*, *description=None*, *default_pool_id=None*, *load_balancer_id=None*, *protocol=None*, *protocol_port=None*, *connection_limit=None*, *enabled=None*, *provisioning_status=None*, *operating_status=None*, *tls_certificate_id=None*, *stats=None*, *default_pool=None*, *load_balancer=None*, *sni_containers=None*, *peer_port=None*, *l7policies=None*, *pools=None*, *insert_headers=None*, *created_at=None*, *updated_at=None*, *timeout_client_data=None*, *timeout_member_connect=None*, *timeout_member_data=None*, *timeout_tcp_inspect=None*, *tags=None*, *client_ca_tls_certificate_id=None*, *client_authentication=None*, *client_crl_container_id=None*, *allowed_cidrs=None*, *tls_ciphers=None*, *tls_versions=None*, *alpn_protocols=None*)

Bases: `octavia.common.data_models.`

delete()

update(*update_dict*)

Generic update method which works for simple,

non-relational attributes.

class **ListenerCidr**(*listener_id=None*, *cidr=None*)

Bases: `octavia.common.data_models.`

to_dict(***kwargs*)

Converts a data model to a dictionary.

class **ListenerStatistics**(*listener_id=None*, *amphora_id=None*, *bytes_in=0*, *bytes_out=0*, *active_connections=0*, *total_connections=0*, *request_errors=0*, *received_time=0.0*)

Bases: `octavia.common.data_models.`

db_fields()

get_stats()

class **LoadBalancer**(*id=None*, *project_id=None*, *name=None*, *description=None*, *provisioning_status=None*, *operating_status=None*, *enabled=None*, *topology=None*, *vip=None*, *listeners=None*, *amphorae=None*, *pools=None*, *vrrp_group=None*, *server_group_id=None*, *created_at=None*, *updated_at=None*, *provider=None*, *tags=None*, *flavor_id=None*, *availability_zone=None*)

Bases: `octavia.common.data_models.`

update(*update_dict*)

Generic update method which works for simple,

non-relational attributes.

class **LoadBalancerStatistics**(*bytes_in=0*, *bytes_out=0*, *active_connections=0*, *total_connections=0*, *request_errors=0*, *listeners=None*)

Bases: `octavia.common.data_models.`

get_stats()

class **Member**(*id=None*, *project_id=None*, *pool_id=None*, *ip_address=None*, *protocol_port=None*, *weight=None*, *backup=None*, *enabled=None*, *subnet_id=None*, *operating_status=None*, *pool=None*, *created_at=None*, *updated_at=None*, *provisioning_status=None*, *name=None*, *monitor_address=None*, *monitor_port=None*, *tags=None*)

Bases: `octavia.common.data_models.`

delete()

class **Pool**(*id=None*, *project_id=None*, *name=None*, *description=None*, *protocol=None*, *lb_algorithm=None*, *enabled=None*, *operating_status=None*, *members=None*, *health_monitor=None*, *session_persistence=None*, *load_balancer_id=None*, *load_balancer=None*, *listeners=None*, *l7policies=None*, *created_at=None*, *updated_at=None*, *provisioning_status=None*, *tags=None*, *tls_certificate_id=None*, *ca_tls_certificate_id=None*, *crl_container_id=None*, *tls_enabled=None*, *tls_ciphers=None*, *tls_versions=None*, *alpn_protocols=None*)

Bases: `octavia.common.data_models.`

delete()

update(*update_dict*)

Generic update method which works for simple,

non-relational attributes.

class **Quotas**(*project_id=None*, *load_balancer=None*, *listener=None*, *pool=None*, *health_monitor=None*, *member=None*, *l7policy=None*, *l7rule=None*, *in_use_health_monitor=None*, *in_use_listener=None*, *in_use_load_balancer=None*, *in_use_member=None*, *in_use_pool=None*, *in_use_l7policy=None*, *in_use_l7rule=None*)

Bases: `octavia.common.data_models.`

class **SNI**(*listener_id=None*, *position=None*, *listener=None*, *tls_container_id=None*)

Bases: `octavia.common.data_models.`

to_dict(***kwargs*)

Converts a data model to a dictionary.

class **SessionPersistence**(*pool_id=None*, *type=None*, *cookie_name=None*, *pool=None*, *persistence_timeout=None*, *persistence_granularity=None*)

Bases: `octavia.common.data_models.`

delete()

class **TLSContainer**(*id=None*, *primary_cn=None*, *certificate=None*, *private_key=None*, *passphrase=None*, *intermediates=None*)

Bases: `octavia.common.data_models.`

class VRRPGroup(*load_balancer_id=None*, *vrrp_group_name=None*, *vrrp_auth_type=None*, *vrrp_auth_pass=None*, *advert_int=None*, *smtp_server=None*, *smtp_connect_timeout=None*, *load_balancer=None*)

Bases: `octavia.common.data_models.`

class Vip(*load_balancer_id=None*, *ip_address=None*, *subnet_id=None*, *network_id=None*, *port_id=None*, *load_balancer=None*, *qos_policy_id=None*, *octavia_owned=None*)

Bases: `octavia.common.data_models.`

octavia.common.decorators module

Decorators to provide backwards compatibility for V1 API.

rename_kwargs(***renamed_kwargs*)

Renames a class's variables and maintains backwards compatibility.

Parameters **renamed_kwargs** -- mapping of old kwargs to new kwargs. For example, to say a class has renamed variable foo to bar the decorator would be used like: rename_kwargs(foo='bar')

octavia.common.exceptions module

Octavia base exception handling.

exception APIException(***kwargs*)

Bases: `webob.exc.HTTPClientError`

code = 500

msg = 'Something unknown went wrong'

exception AmphoraNetworkConfigException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Cannot configure network resource in the amphora: %(detail)s'

exception CertificateGenerationException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Could not sign the certificate request: %(msg)s'

exception CertificateRetrievalException(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'Could not retrieve certificate: %(ref)s'

exception CertificateStorageException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Could not store certificate: %(msg)s'

exception ComputeBuildException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to build compute instance due to: %(fault)s'

exception ComputeBuildQueueTimeoutException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to get an amphora build slot.'

exception ComputeDeleteException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to delete compute instance. The compute service reports: %(compute_msg)s'

exception ComputeGetException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to retrieve compute instance.'

exception ComputeGetInterfaceException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to retrieve compute virtual interfaces.'

exception ComputePortInUseException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Compute driver reports port %(port)s is already in use.'

exception ComputeStatusException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to retrieve compute instance status.'

exception ComputeUnknownException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Unknown exception from the compute driver: %(exc)s.'

exception ComputeWaitTimeoutException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

message = 'Waiting for compute id %(id)s to go active timeout.'

exception DisabledOption(***kwargs*)
Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'The selected %(option)s is not allowed in this deployment: %(value)s'

exception DuplicateHealthMonitor(***kwargs*)
Bases: *octavia.common.exceptions.APIException*

code = 409

msg = 'This pool already has a health monitor'

exception DuplicateListenerEntry(***kwargs*)
Bases: *octavia.common.exceptions.APIException*

code = 409

msg = 'Another Listener on this Load Balancer is already using protocol %(protocol)s and protocol_port %(port)d'

exception DuplicateMemberEntry(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 409`

`msg = 'Another member on this pool is already using ip %(ip_address)s on protocol_port %(port)d'`

exception DuplicatePoolEntry(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 409`

`msg = 'This listener already has a default pool'`

exception IDAlreadyExists(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 409`

`msg = 'Already an entity with that specified id.'`

exception ImageGetException(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Failed to retrieve image with %(tag)s tag.'`

exception ImmutableObject(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 409`

`msg = '%(resource)s %(id)s is immutable and cannot be updated.'`

exception InputFileError(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Error with file %(file_name)s. Reason:  %(reason)s'`

exception InvalidAmphoraOperatingSystem(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Invalid amphora operating system:  %(os_name)s'`

exception InvalidFilterArgument(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 400`

`msg = 'One or more arguments are either duplicate or invalid'`

exception InvalidHMACException(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = "HMAC hashes didn't match"`

exception InvalidIPAddress(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 400`

`msg = 'The IP Address %(ip_addr)s is invalid.'`

exception InvalidL7PolicyAction(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'Invalid L7 Policy action specified: %(action)s'

exception InvalidL7PolicyArgs(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'Invalid L7 Policy arguments: %(msg)s'

exception InvalidL7Rule(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Invalid L7 Rule: %(msg)s'

exception InvalidLimit(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = "Supplied pagination limit '%(key)s' is not valid."

exception InvalidMarker(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = "Supplied pagination marker '%(key)s' is not valid."

exception InvalidOption(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = '%(value)s is not a valid option for %(option)s'

exception InvalidRegex(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Unable to parse regular expression: %(e)s'

exception InvalidSortDirection(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = "Supplied sort direction '%(key)s' is not valid."

exception InvalidSortKey(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = "Supplied sort key '%(key)s' is not valid."

exception InvalidString(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Invalid characters in %(what)s'

exception InvalidSubresource(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 400`

`msg = '%(resource)s %(id)s not found.'`

exception InvalidTopology(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Invalid topology specified:  %(topology)s'`

exception InvalidURL(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Not a valid URL: %(url)s'`

exception InvalidURLPath(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 400`

`msg = 'Not a valid URLPath:  %(url_path)s'`

exception L7RuleValidation(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 400`

`msg = 'Error parsing L7Rule:  %(error)s'`

exception LBPendingStateError(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 409`

`msg = 'Invalid state %(state)s of loadbalancer resource %(id)s'`

exception MisMatchedKey(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Key and x509 certificate do not match'`

exception MissingAPIProjectID(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

`code = 400`

`message = 'Missing project ID in request where one is required.'`

exception MissingArguments(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Missing arguments.'`

exception MissingProjectID(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'Missing project ID in request where one is required.'`

exception MissingVIPSecurityGroup(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

`message = 'VIP security group is missing for load balancer:  %(lb_id)s'`

exception NeedsPassphrase(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

message = 'Passphrase needed to decrypt key but client did not provide one.'

exception NetworkConfig(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

message = 'Unable to allocate network resource from config'

exception NetworkServiceError(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

message = 'The networking service had a failure: %(net_error)s'

exception NoReadyAmphoraeException(**args*, ***kwargs*)

Bases: `octavia.common.exceptions.OctaviaException`

message = 'There are not any READY amphora available.'

exception NotFound(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

code = 404

msg = '%(resource)s %(id)s not found.'

exception ObjectInUse(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

code = 409

msg = '%(object)s %(id)s is in use and cannot be modified.'

exception OctaviaException(**args*, ***kwargs*)

Bases: `Exception`

Base Octavia Exception.

To correctly use this class, inherit from it and define a 'message' property. That message will get printf'd with the keyword arguments provided to the constructor.

message = 'An unknown exception occurred.'

orig_code = None

orig_msg = None

static use_fatal_exceptions()

exception PolicyForbidden(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

code = 403

msg = 'Policy does not allow this request to be performed.'

exception PoolInUseByL7Policy(***kwargs*)

Bases: `octavia.common.exceptions.APIException`

code = 409

msg = 'Pool %(id)s is in use by L7 policy %(l7policy_id)s'

exception ProjectBusyException(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 503`

`msg = 'Project busy. Unable to lock the project. Please try again.'`

exception ProviderDriverError(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 500`

`msg = "Provider '%(prov)s' reports error:  %(user_msg)s"`

exception ProviderFlavorMismatchError(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 400`

`msg = "Flavor '%(flav)s' is not compatible with provider '%(prov)s'"`

exception ProviderNotEnabled(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 400`

`msg = "Provider '%(prov)s' is not enabled."`

exception ProviderNotFound(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 501`

`msg = "Provider '%(prov)s' was not found."`

exception ProviderNotImplementedError(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 501`

`msg = "Provider '%(prov)s' does not support a requested action: %(user_msg)s"`

exception ProviderUnsupportedOptionError(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 501`

`msg = "Provider '%(prov)s' does not support a requested option: %(user_msg)s"`

exception QuotaException(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 403`

`msg = 'Quota has been met for resources:  %(resource)s'`

exception RecordAlreadyExists(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

`code = 409`

`msg = 'A %(field)s of %(name)s already exists.'`

exception ServerGroupObjectCreateException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to create server group object.'

exception ServerGroupObjectDeleteException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to delete server group object.'

exception SingleCreateDetailsMissing(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'Missing details for %(type)s object: %(name)s'

exception TooManyL7RulesOnL7Policy(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 409

msg = 'Too many rules on L7 policy %(id)s'

exception UnreadableCert(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Could not read X509 from PEM'

exception UnreadablePKCS12(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'The PKCS12 bundle is unreadable. Please check the PKCS12 bundle validity. In addition, make sure it does not require a pass phrase. Error: %(error)s'

exception VIPValidationException(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'Validation failure: VIP must contain one of: %(objects)s.'

exception ValidationException(***kwargs*)

Bases: *octavia.common.exceptions.APIException*

code = 400

msg = 'Validation failure: %(detail)s'

exception VolumeDeleteException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to delete volume instance.'

exception VolumeGetException(**args*, ***kwargs*)

Bases: *octavia.common.exceptions.OctaviaException*

message = 'Failed to retrieve volume instance.'

octavia.common.keystone module

class KeystoneSession(*section='service_auth'*)

Bases: `object`

get_auth()

get_service_user_id()

get_session()

Initializes a Keystone session.

Returns a Keystone Session object

class SkippingAuthProtocol(*app*, *conf*)

Bases: `keystonemiddleware.auth_token.AuthProtocol`

SkippingAuthProtocol to reach special endpoints

Bypasses keystone authentication for special request paths, such as the api version discovery path.

Note: SkippingAuthProtocol is lean customization of `keystonemiddleware.auth_token.AuthProtocol` that disables keystone communication if the request path is in the _NOAUTH_PATHS list.

process_request(*request*)

Process request.

Evaluate the headers in a request and attempt to authenticate the request. If authenticated then additional headers are added to the request for use by applications. If not authenticated the request will be rejected or marked unauthenticated depending on configuration.

octavia.common.policy module

Policy Engine For Octavia.

class IsAdminCheck(*kind*, *match*)

Bases: `oslo_policy._checks.Check`

An explicit check for is_admin.

class Policy(*conf=<oslo_config.cfg.ConfigOpts object>*, *policy_file=None*, *rules=None*, *default_rule=None*, *use_conf=True*, *overwrite=True*)

Bases: `oslo_policy.policy.Enforcer`

authorize(*action*, *target*, *context*, *do_raise=True*, *exc=None*)

Verifies that the action is valid on the target in this context.

Parameters

- **context** -- The oslo context for this request.
- **action** -- string representing the action to be checked this should be colon separated for clarity. i.e. `compute:create_instance`, `compute:attach_volume`, `volume:attach_volume`
- **target** -- dictionary representing the object of the action for object creation this should be a dictionary representing the location of the object e.g. `{'project_id': context.project_id}`

- **do_raise** -- if True (the default), raises PolicyForbidden; if False, returns False
- **exc** -- Class of the exceptions to raise if the check fails. Any remaining arguments passed to `enforce()` (both positional and keyword arguments) will be passed to the exceptions class. If not specified, `PolicyForbidden` will be used.

Raises *PolicyForbidden* -- if verification fails and do_raise is True. Or if 'exc' is specified it will raise an exceptions of that type.

Returns returns a non-False value (not necessarily "True") if authorized, and the exact value False if not authorized and do_raise is False.

check_is_admin(*context*)

Does roles contains 'admin' role according to policy setting.

get_rules()

get_enforcer()

get_no_context_enforcer()

reset()

octavia.common.rpc module

cleanup()

create_transport(*url*)

get_client(*target*, *version_cap=None*, *serializer=None*, *call_monitor_timeout=None*)

get_server(*target*, *endpoints*, *executor='threading'*, *access_policy=<class 'oslo_messaging.rpc.dispatcher.DefaultRPCAccessPolicy'>*, *serializer=None*)

get_transport_url(*url_str=None*)

init()

octavia.common.service module

prepare_service(*argv=None*)

Sets global config from config file and sets up logging.

octavia.common.stats module

class StatsMixin

Bases: `object`

get_listener_stats(*session*, *listener_id*)

Gets the listener statistics data_models object.

get_loadbalancer_stats(*session*, *loadbalancer_id*)

octavia.common.utils module

Utilities and helper functions.

`b`(*s*)

`base64_sha1_string`(*string_to_hash*)
Get a b64-encoded sha1 hash of a string. Not intended to be secure!

`class` **`exception_logger`**(*logger=None*)
Bases: `object`

Wrap a function and log raised exception

Parameters **`logger`** -- the logger to log the exception default is LOG.exception

Returns origin value if no exception raised; re-raise the exception if any occurred

`expand_expected_codes`(*codes*)
Expand the expected code string in set of codes.

200-204 -> 200, 201, 202, 204 200, 203 -> 200, 203

`get_amphora_driver`()

`get_compatible_server_certs_key_passphrase`()

`get_compatible_value`(*value*)

`get_hostname`()

`get_network_driver`()

`get_vip_security_group_name`(*loadbalancer_id*)

`ip_netmask_to_cidr`(*ip*, *netmask*)

`ip_port_str`(*ip_address*, *port*)
Return IP port as string representation depending on address family.

`is_cidr_ipv6`(*cidr*)
Check if CIDR is IPv6 address with subnet prefix.

`is_ipv4`(*ip_address*)
Check if ip address is IPv4 address.

`is_ipv6`(*ip_address*)
Check if ip address is IPv6 address.

`is_ipv6_lla`(*ip_address*)
Check if ip address is IPv6 link local address.

`netmask_to_prefix`(*netmask*)

`subnet_ip_availability`(*nw_ip_avail*, *subnet_id*, *req_num_ips*)

octavia.common.validate module

Several handy validation functions that go beyond simple type checking. Defined here so these can also be used at deeper levels than the API.

check_alpn_protocols(*protocols*)

check_cipher_prohibit_list(*cipherstring*)

check_default_ciphers_prohibit_list_conflict()

check_default_tls_versions_min_conflict()

check_port_in_use(*port*)

Raise an exception when a port is used.

check_session_persistence(*SP_dict*)

check_tls_version_list(*versions*)

check_tls_version_min(*versions*, *message=None*)

Checks a TLS version string against the configured minimum.

cookie_value_string(*value*, *what=None*)

Raises an error if the value string contains invalid characters.

header_name(*header*, *what=None*)

Raises an error if header does not look like an HTML header name.

header_value_string(*value*, *what=None*)

Raises an error if the value string contains invalid characters.

ip_not_reserved(*ip_address*)

is_ip_member_of_cidr(*address*, *cidr*)

l7rule_data(*l7rule*)

Raises an error if the l7rule given is invalid in some way.

network_allowed_by_config(*network_id*, *valid_networks=None*)

network_exists_optionally_contains_subnet(*network_id*, *subnet_id=None*, *context=None*)

Raises an exception when a network does not exist.

If a subnet is provided, also validate the network contains that subnet.

port_exists(*port_id*, *context=None*)

Raises an exception when a port does not exist.

qos_extension_enabled(*network_driver*)

qos_policy_exists(*qos_policy_id*)

regex(*regex*)

Raises an error if the string given is not a valid regex.

sanitize_l7policy_api_args(*l7policy*, *create=False*)

Validate and make consistent L7Policy API arguments.

This method is mainly meant to sanitize L7 Policy create and update API dictionaries, so that we strip 'None' values that don't apply for our particular update. This method does *not* verify that any redirect_pool_id exists in the database, but will raise an error if a redirect_url doesn't look like a URL.

Parameters **`l7policy`** -- The L7 Policy dictionary we are santizing / validating

`subnet_exists`(*subnet_id*, *context=None*)
Raises an exception when a subnet does not exist.

`url`(*url*, *require_scheme=True*)
Raises an error if the url doesn't look like a URL.

`url_path`(*url_path*)
Raises an error if the url_path doesn't look like a URL Path.

`validate_l7rule_ssl_types`(*l7rule*)

Module contents

octavia.compute package

Subpackages

octavia.compute.drivers package

Subpackages

octavia.compute.drivers.noop_driver package

Submodules

octavia.compute.drivers.noop_driver.driver module

`class` **`NoopComputeDriver`**
Bases: `octavia.compute.compute_base.ComputeBase`

`attach_network_or_port`(*compute_id*, *network_id=None*, *ip_address=None*, *port_id=None*)
Connects an existing amphora to an existing network.

Parameters

- **`compute_id`** -- id of an amphora in the compute service
- **`network_id`** -- id of a network
- **`ip_address`** -- ip address to attempt to be assigned to interface
- **`port_id`** -- id of the neutron port

Returns nova interface

Raises Exception

build(*name='amphora_name'*, *amphora_flavor=None*, *image_tag=None*, *image_owner=None*, *key_name=None*, *sec_groups=None*, *network_ids=None*, *config_drive_files=None*, *user_data=None*, *port_ids=None*, *server_group_id=None*, *availability_zone=None*)

Build a new amphora.

Parameters

- **name** -- Optional name for Amphora
- **amphora_flavor** -- Optionally specify a flavor
- **image_tag** -- tag of the base image for the amphora instance
- **key_name** -- Optionally specify a keypair
- **sec_groups** -- Optionally specify list of security groups
- **network_ids** -- A list of network IDs to attach to the amphora
- **config_drive_files** -- An optional dict of files to overwrite on the server upon boot. Keys are file names (i.e. /etc/passwd) and values are the file contents (either as a string or as a file-like object). A maximum of five entries is allowed, and each file must be 10k or less.
- **user_data** -- Optional user data to pass to be exposed by the metadata server this can be a file type object as well or a string
- **server_group_id** -- Optional server group id(uuid) which is used for anti_affinity feature
- **availability_zone** -- Name of the compute availability zone.

Raises *ComputeBuildException* -- if compute failed to build amphora

Returns UUID of amphora

create_server_group(*name*, *policy*)

Create a server group object

Parameters

- **name** -- the name of the server group
- **policy** -- the policy of the server group

Returns the server group object

delete(*compute_id*)

Delete the specified amphora

Parameters **compute_id** -- The id of the amphora to delete

delete_server_group(*server_group_id*)

Delete a server group object

Parameters **server_group_id** -- the uuid of a server group

detach_port(*compute_id*, *port_id*)

Disconnects an existing amphora from an existing port.

Parameters

- **compute_id** -- id of an amphora in the compute service

- **`port_id`** -- id of the port

Returns None

Raises Exception

`get_amphora`(*compute_id*, *management_network_id=None*)
Retrieve an amphora object

Parameters

- **`compute_id`** -- the compute id of the desired amphora
- **`management_network_id`** -- ID of the management network

Returns the amphora object

Returns fault message or None

`status`(*compute_id*)
Check whether the specified amphora is up

Parameters **`compute_id`** -- the ID of the desired amphora

Returns The compute "status" response ("ONLINE" or "OFFLINE")

`validate_availability_zone`(*availability_zone*)
Validates that a compute availability zone exists.

Parameters **`availability_zone`** -- Name of the compute availability zone.

Returns None

Raises NotFound

Raises NotImplementedError

`validate_flavor`(*flavor_id*)
Validates that a compute flavor exists.

Parameters **`flavor_id`** -- ID of the compute flavor.

Returns None

Raises NotFound

Raises NotImplementedError

`class` **`NoopManager`**
Bases: `object`

`attach_network_or_port`(*compute_id*, *network_id=None*, *ip_address=None*, *port_id=None*)

`build`(*name='amphora_name'*, *amphora_flavor=None*, *image_tag=None*, *image_owner=None*, *key_name=None*, *sec_groups=None*, *network_ids=None*, *config_drive_files=None*, *user_data=None*, *port_ids=None*, *server_group_id=None*, *availability_zone=None*)

`create_server_group`(*name*, *policy*)

`delete`(*compute_id*)

`delete_server_group`(*server_group_id*)

`detach_port`(*compute_id*, *port_id*)

`get_amphora`(*compute_id*, *management_network_id=None*)

`status`(*compute_id*)

`validate_availability_zone`(*availability_zone*)

`validate_flavor`(*flavor_id*)

`NoopServerGroup`

alias of `octavia.compute.drivers.noop_driver.driver.ServerGroup`

Module contents

Submodules

octavia.compute.drivers.nova_driver module

`class VirtualMachineManager`

Bases: *`octavia.compute.compute_base.ComputeBase`*

Compute implementation of virtual machines via nova.

`attach_network_or_port`(*compute_id*, *network_id=None*, *ip_address=None*, *port_id=None*)

Attaching a port or a network to an existing amphora

Parameters

- **`compute_id`** -- id of an amphora in the compute service
- **`network_id`** -- id of a network
- **`ip_address`** -- ip address to attempt to be assigned to interface
- **`port_id`** -- id of the neutron port

Returns nova interface instance

Raises

- *`ComputePortInUseException`* -- The port is in use somewhere else
- *`ComputeUnknownException`* -- Unknown nova error

`build`(*name='amphora_name'*, *amphora_flavor=None*, *image_tag=None*, *image_owner=None*, *key_name=None*, *sec_groups=None*, *network_ids=None*, *port_ids=None*, *config_drive_files=None*, *user_data=None*, *server_group_id=None*, *availability_zone=None*)

Create a new virtual machine.

Parameters

- **`name`** -- optional name for amphora
- **`amphora_flavor`** -- image flavor for virtual machine
- **`image_tag`** -- image tag for virtual machine
- **`key_name`** -- keypair to add to the virtual machine
- **`sec_groups`** -- Security group IDs for virtual machine
- **`network_ids`** -- Network IDs to include on virtual machine

- **port_ids** -- Port IDs to include on virtual machine
- **config_drive_files** -- An optional dict of files to overwrite on the server upon boot. Keys are file names (i.e. /etc/passwd) and values are the file contents (either as a string or as a file-like object). A maximum of five entries is allowed, and each file must be 10k or less.
- **user_data** -- Optional user data to pass to be exposed by the metadata server this can be a file type object as well or a string
- **server_group_id** -- Optional server group id(uuid) which is used for anti_affinity feature
- **availability_zone** -- Name of the compute availability zone.

Raises *ComputeBuildException* -- if nova failed to build virtual machine

Returns UUID of amphora

create_server_group(*name*, *policy*)

Create a server group object

Parameters

- **name** -- the name of the server group
- **policy** -- the policy of the server group

Raises Generic exception if the server group is not created

Returns the server group object

delete(*compute_id*)

Delete a virtual machine.

Parameters **compute_id** -- virtual machine UUID

delete_server_group(*server_group_id*)

Delete a server group object

Raises Generic exception if the server group is not deleted

Parameters **server_group_id** -- the uuid of a server group

detach_port(*compute_id*, *port_id*)

Detaches a port from an existing amphora.

Parameters

- **compute_id** -- id of an amphora in the compute service
- **port_id** -- id of the port

Returns None

get_amphora(*compute_id*, *management_network_id=None*)

Retrieve the information in nova of a virtual machine.

Parameters

- **compute_id** -- virtual machine UUID
- **management_network_id** -- ID of the management network

Returns an amphora object

Returns fault message or None

`status`(*compute_id*)
Retrieve the status of a virtual machine.

Parameters **`compute_id`** -- virtual machine UUID

Returns constant of amphora status

`validate_availability_zone`(*availability_zone*)
Validates that an availability zone exists in nova.

Parameters **`availability_zone`** -- Name of the availability zone to lookup.

Raises NotFound

Returns None

`validate_flavor`(*flavor_id*)
Validates that a flavor exists in nova.

Parameters **`flavor_id`** -- ID of the flavor to lookup in nova.

Raises NotFound

Returns None

Module contents

Submodules

octavia.compute.compute_base module

`class ComputeBase`
Bases: `object`

`abstract attach_network_or_port`(*compute_id*, *network_id=None*, *ip_address=None*, *port_id=None*)
Connects an existing amphora to an existing network.

Parameters

- **`compute_id`** -- id of an amphora in the compute service
- **`network_id`** -- id of a network
- **`ip_address`** -- ip address to attempt to be assigned to interface
- **`port_id`** -- id of the neutron port

Returns nova interface

Raises Exception

`abstract build`(*name='amphora_name'*, *amphora_flavor=None*, *image_tag=None*, *image_owner=None*, *key_name=None*, *sec_groups=None*, *network_ids=None*, *config_drive_files=None*, *user_data=None*, *server_group_id=None*, *availability_zone=None*)
Build a new amphora.

Parameters

- **`name`** -- Optional name for Amphora
- **`amphora_flavor`** -- Optionally specify a flavor
- **`image_tag`** -- tag of the base image for the amphora instance
- **`key_name`** -- Optionally specify a keypair
- **`sec_groups`** -- Optionally specify list of security groups
- **`network_ids`** -- A list of network IDs to attach to the amphora
- **`config_drive_files`** -- An optional dict of files to overwrite on the server upon boot. Keys are file names (i.e. /etc/passwd) and values are the file contents (either as a string or as a file-like object). A maximum of five entries is allowed, and each file must be 10k or less.
- **`user_data`** -- Optional user data to pass to be exposed by the metadata server this can be a file type object as well or a string
- **`server_group_id`** -- Optional server group id(uuid) which is used for anti_affinity feature
- **`availability_zone`** -- Name of the compute availability zone.

Raises *`ComputeBuildException`* -- if compute failed to build amphora

Returns UUID of amphora

`abstract create_server_group`(*name*, *policy*)

Create a server group object

Parameters

- **`name`** -- the name of the server group
- **`policy`** -- the policy of the server group

Returns the server group object

`abstract delete`(*compute_id*)

Delete the specified amphora

Parameters **`compute_id`** -- The id of the amphora to delete

`abstract delete_server_group`(*server_group_id*)

Delete a server group object

Parameters **`server_group_id`** -- the uuid of a server group

`abstract detach_port`(*compute_id*, *port_id*)

Disconnects an existing amphora from an existing port.

Parameters

- **`compute_id`** -- id of an amphora in the compute service
- **`port_id`** -- id of the port

Returns None

Raises Exception

abstract get_amphora(*compute_id*, *management_network_id=None*)
Retrieve an amphora object

Parameters

- **compute_id** -- the compute id of the desired amphora
- **management_network_id** -- ID of the management network

Returns the amphora object

Returns fault message or None

abstract status(*compute_id*)
Check whether the specified amphora is up

Parameters **compute_id** -- the ID of the desired amphora

Returns The compute "status" response ("ONLINE" or "OFFLINE")

abstract validate_availability_zone(*availability_zone*)
Validates that a compute availability zone exists.

Parameters **availability_zone** -- Name of the compute availability zone.

Returns None

Raises NotFound

Raises NotImplementedError

abstract validate_flavor(*flavor_id*)
Validates that a compute flavor exists.

Parameters **flavor_id** -- ID of the compute flavor.

Returns None

Raises NotFound

Raises NotImplementedError

Module contents

octavia.controller package

Subpackages

octavia.controller.healthmanager package

Submodules

octavia.controller.healthmanager.health_manager module

class HealthManager(*exit_event*)
Bases: `object`

health_check()

update_stats_on_done(*stats*, *fut*)

wait_done_or_dead(*futs*, *dead*, *check_timeout=1*)

Module contents

octavia.controller.housekeeping package

Submodules

octavia.controller.housekeeping.house_keeping module

class CertRotation

Bases: `object`

rotate()

Check the amphora db table for expiring auth certs.

class DatabaseCleanup

Bases: `object`

cleanup_load_balancers()

Checks the DB for old load balancers and triggers their removal.

delete_old_amphorae()

Checks the DB for old amphora and deletes them based on its age.

Module contents

octavia.controller.queue package

Subpackages

octavia.controller.queue.v1 package

Submodules

octavia.controller.queue.v1.consumer module

class ConsumerService(*worker_id*, *conf*)

Bases: `cotyledon._service.Service`

run()

Method representing the service activity

If not implemented the process will just wait to receive an ending signal.

This method is ran into the thread and can block or return as needed

Any exceptions raised by this method will be logged and the worker will exit with status 1.

terminate()

Gracefully shutdown the service

This method will be executed when the Service has to shutdown cleanly.

If not implemented the process will just end with status 0.

To customize the exit code, the `SystemExit` exception can be used.

Any exceptions raised by this method will be logged and the worker will exit with status 1.

octavia.controller.queue.v1.endpoints module

class **Endpoints**

Bases: `object`

batch_update_members(*context*, *old_member_ids*, *new_member_ids*, *updated_members*)

create_health_monitor(*context*, *health_monitor_id*)

create_l7policy(*context*, *l7policy_id*)

create_l7rule(*context*, *l7rule_id*)

create_listener(*context*, *listener_id*)

create_load_balancer(*context*, *load_balancer_id*, *flavor=None*, *availability_zone=None*)

create_member(*context*, *member_id*)

create_pool(*context*, *pool_id*)

delete_amphora(*context*, *amphora_id*)

delete_health_monitor(*context*, *health_monitor_id*)

delete_l7policy(*context*, *l7policy_id*)

delete_l7rule(*context*, *l7rule_id*)

delete_listener(*context*, *listener_id*)

delete_load_balancer(*context*, *load_balancer_id*, *cascade=False*)

delete_member(*context*, *member_id*)

delete_pool(*context*, *pool_id*)

failover_amphora(*context*, *amphora_id*)

failover_load_balancer(*context*, *load_balancer_id*)

target = <Target namespace=controller, version=1.1>

update_amphora_agent_config(*context*, *amphora_id*)

update_health_monitor(*context*, *health_monitor_id*, *health_monitor_updates*)

update_l7policy(*context*, *l7policy_id*, *l7policy_updates*)

update_l7rule(*context*, *l7rule_id*, *l7rule_updates*)

update_listener(*context*, *listener_id*, *listener_updates*)

update_load_balancer(*context*, *load_balancer_id*, *load_balancer_updates*)

update_member(*context*, *member_id*, *member_updates*)

update_pool(*context*, *pool_id*, *pool_updates*)

Module contents

octavia.controller.queue.v2 package

Submodules

octavia.controller.queue.v2.consumer module

class ConsumerService(*worker_id*, *conf*)

Bases: `cotyledon._service.Service`

run()

Method representing the service activity

If not implemented the process will just wait to receive an ending signal.

This method is ran into the thread and can block or return as needed

Any exceptions raised by this method will be logged and the worker will exit with status 1.

terminate()

Gracefully shutdown the service

This method will be executed when the Service has to shutdown cleanly.

If not implemented the process will just end with status 0.

To customize the exit code, the `SystemExit` exception can be used.

Any exceptions raised by this method will be logged and the worker will exit with status 1.

octavia.controller.queue.v2.endpoints module

class Endpoints

Bases: `object`

batch_update_members(*context*, *old_members*, *new_members*, *updated_members*)

create_health_monitor(*context*, *health_monitor*)

create_l7policy(*context*, *l7policy*)

create_l7rule(*context*, *l7rule*)

create_listener(*context*, *listener*)

create_load_balancer(*context*, *loadbalancer*, *flavor=None*, *availability_zone=None*)

create_member(*context*, *member*)

create_pool(*context*, *pool*)

delete_amphora(*context*, *amphora_id*)

delete_health_monitor(*context*, *health_monitor*)

delete_l7policy(*context*, *l7policy*)

delete_l7rule(*context*, *l7rule*)

delete_listener(*context*, *listener*)

delete_load_balancer(*context*, *loadbalancer*, *cascade=False*)

delete_member(*context*, *member*)

delete_pool(*context*, *pool*)

failover_amphora(*context*, *amphora_id*)

failover_load_balancer(*context*, *load_balancer_id*)

target = <Target namespace=controller, version=2.0>

update_amphora_agent_config(*context*, *amphora_id*)

update_health_monitor(*context*, *original_health_monitor*, *health_monitor_updates*)

update_l7policy(*context*, *original_l7policy*, *l7policy_updates*)

update_l7rule(*context*, *original_l7rule*, *l7rule_updates*)

update_listener(*context*, *original_listener*, *listener_updates*)

update_load_balancer(*context*, *original_load_balancer*, *load_balancer_updates*)

update_member(*context*, *original_member*, *member_updates*)

update_pool(*context*, *original_pool*, *pool_updates*)

Module contents

Module contents

octavia.controller.worker package

Subpackages

octavia.controller.worker.v1 package

Subpackages

octavia.controller.worker.v1.flows package

Submodules

octavia.controller.worker.v1.flows.amphora_flows module

class **AmphoraFlows**

Bases: `object`

`cert_rotate_amphora_flow`()

Implement rotation for amphora's cert.

1. Create a new certificate
2. Upload the cert to amphora
3. update the newly created certificate info to amphora
4. update the cert_busy flag to be false after rotation

Returns The flow for updating an amphora

`get_amphora_for_lb_failover_subflow`(*prefix*, *role='STANDALONE'*, *failed_amp_vrrp_port_id=None*, *is_vrrp_ipv6=False*)

Creates a new amphora that will be used in a failover flow.

Requires loadbalancer_id, flavor, vip, vip_sg_id, loadbalancer

Provides amphora_id, amphora

Parameters

- **`prefix`** -- The flow name prefix to use on the flow and tasks.
- **`role`** -- The role this amphora will have in the topology.
- **`failed_amp_vrrp_port_id`** -- The base port ID of the failed amp.
- **`is_vrrp_ipv6`** -- True if the base port IP is IPv6.

Returns A Taskflow sub-flow that will create the amphora.

`get_amphora_for_lb_subflow`(*prefix*, *role='STANDALONE'*)

`get_create_amphora_flow`()

Creates a flow to create an amphora.

Returns The flow for creating the amphora

`get_delete_amphora_flow`(*amphora*, *retry_attempts=5*, *retry_interval=5*)

Creates a subflow to delete an amphora and it's port.

This flow is idempotent and safe to retry.

Parameters

- **`amphora`** -- An amphora object.
- **`retry_attempts`** -- The number of times the flow is retried.
- **`retry_interval`** -- The time to wait, in seconds, between retries.

Returns The subflow for deleting the amphora.

Raises *`AmphoraNotFound`* -- The referenced Amphora was not found.

`get_failover_amphora_flow`(*failed_amphora*, *lb_amp_count*)

Get a Taskflow flow to failover an amphora.

1. Build a replacement amphora.
2. Delete the old amphora.

3. Update the amphorae listener configurations.
4. Update the VRRP configurations if needed.

Parameters

- **`failed_amphora`** -- The amphora object to failover.
- **`lb_amp_count`** -- The number of amphora on this load balancer.

Returns The flow that will provide the failover.

`get_vrrp_subflow`(*prefix*, *timeout_dict=None*, *create_vrrp_group=True*, *get_amphorae_status=True*)

`update_amphora_config_flow`()

Creates a flow to update the amphora agent configuration.

Returns The flow for updating an amphora

octavia.controller.worker.v1.flows.health_monitor_flows module

class `HealthMonitorFlows`

Bases: `object`

`get_create_health_monitor_flow`()

Create a flow to create a health monitor

Returns The flow for creating a health monitor

`get_delete_health_monitor_flow`()

Create a flow to delete a health monitor

Returns The flow for deleting a health monitor

`get_update_health_monitor_flow`()

Create a flow to update a health monitor

Returns The flow for updating a health monitor

octavia.controller.worker.v1.flows.l7policy_flows module

class `L7PolicyFlows`

Bases: `object`

`get_create_l7policy_flow`()

Create a flow to create an L7 policy

Returns The flow for creating an L7 policy

`get_delete_l7policy_flow`()

Create a flow to delete an L7 policy

Returns The flow for deleting an L7 policy

`get_update_l7policy_flow`()

Create a flow to update an L7 policy

Returns The flow for updating an L7 policy

octavia.controller.worker.v1.flows.l7rule_flows module

class L7RuleFlows

Bases: `object`

get_create_l7rule_flow()

Create a flow to create an L7 rule

Returns The flow for creating an L7 rule

get_delete_l7rule_flow()

Create a flow to delete an L7 rule

Returns The flow for deleting an L7 rule

get_update_l7rule_flow()

Create a flow to update an L7 rule

Returns The flow for updating an L7 rule

octavia.controller.worker.v1.flows.listener_flows module

class ListenerFlows

Bases: `object`

get_create_all_listeners_flow()

Create a flow to create all listeners

Returns The flow for creating all listeners

get_create_listener_flow()

Create a flow to create a listener

Returns The flow for creating a listener

get_delete_listener_flow()

Create a flow to delete a listener

Returns The flow for deleting a listener

get_delete_listener_internal_flow(*listener_name*)

Create a flow to delete a listener and l7policies internally

(will skip deletion on the amp and marking LB active)

Returns The flow for deleting a listener

get_update_listener_flow()

Create a flow to update a listener

Returns The flow for updating a listener

octavia.controller.worker.v1.flows.load_balancer_flows module

class LoadBalancerFlows

Bases: `object`

get_cascade_delete_load_balancer_flow(*lb*)

Creates a flow to delete a load balancer.

Returns The flow for deleting a load balancer

get_create_load_balancer_flow(*topology*, *listeners=None*)

Creates a conditional graph flow that allocates a loadbalancer.

Raises *InvalidTopology* -- Invalid topology specified

Returns The graph flow for creating a loadbalancer.

get_delete_load_balancer_flow(*lb*)

Creates a flow to delete a load balancer.

Returns The flow for deleting a load balancer

get_failover_LB_flow(*amps*, *lb*)

Failover a load balancer.

1. Validate the VIP port is correct and present.
2. Build a replacement amphora.
3. Delete the failed amphora.
4. Configure the replacement amphora listeners.
5. Configure VRRP for the listeners.
6. Build the second replacement amphora.
7. Delete the second failed amphora.
8. Delete any extraneous amphora.
9. Configure the listeners on the new amphorae.
10. Configure the VRRP on the new amphorae.
11. Reload the listener configurations to pick up VRRP changes.
12. Mark the load balancer back to ACTIVE.

Returns The flow that will provide the failover.

get_post_lb_amp_association_flow(*prefix*, *topology*)

Reload the loadbalancer and create networking subflows for

created/allocated amphorae. :return: Post amphorae association subflow

get_update_load_balancer_flow()

Creates a flow to update a load balancer.

Returns The flow for update a load balancer

octavia.controller.worker.v1.flows.member_flows module

class MemberFlows

Bases: `object`

get_batch_update_members_flow(*old_members*, *new_members*, *updated_members*)

Create a flow to batch update members

Returns The flow for batch updating members

get_create_member_flow()

Create a flow to create a member

Returns The flow for creating a member

get_delete_member_flow()

Create a flow to delete a member

Returns The flow for deleting a member

get_update_member_flow()

Create a flow to update a member

Returns The flow for updating a member

octavia.controller.worker.v1.flows.pool_flows module

class PoolFlows

Bases: `object`

get_create_pool_flow()

Create a flow to create a pool

Returns The flow for creating a pool

get_delete_pool_flow()

Create a flow to delete a pool

Returns The flow for deleting a pool

get_delete_pool_flow_internal(*name*)

Create a flow to delete a pool, etc.

Returns The flow for deleting a pool

get_update_pool_flow()

Create a flow to update a pool

Returns The flow for updating a pool

Module contents

octavia.controller.worker.v1.tasks package

Submodules

octavia.controller.worker.v1.tasks.amphora_driver_tasks module

`class` **`AmpListenersUpdate`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to update the listeners on one amphora.

`execute``(`*loadbalancer*, *amphora*, *timeout_dict=None*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`AmphoraCertUpload`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Upload a certificate to the amphora.

`execute``(`*amphora*, *server_pem*`)`

Execute cert_update_amphora routine.

`class` **`AmphoraComputeConnectivityWait`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to wait for the compute instance to be up.

`execute``(`*amphora*`)`

Execute get_info routine for an amphora until it responds.

`class` **`AmphoraConfigUpdate`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to push a new amphora agent configuration to the amphora.

execute(*amphora*, *flavor*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **AmphoraFinalize**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to finalize the amphora before any listeners are configured.

execute(*amphora*)

Execute finalize_amphora routine.

revert(*result*, *amphora*, **args*, ***kwargs*)

Handle a failed amphora finalize.

class **AmphoraGetDiagnostics**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to get diagnostics on the amphora and the loadbalancers.

execute(*amphora*)

Execute get_diagnostic routine for an amphora.

class **AmphoraGetInfo**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to get information on an amphora.

execute(*amphora*)

Execute get_info routine for an amphora.

class **AmphoraIndexListenerUpdate**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to update the listeners on one amphora.

execute(*loadbalancer*, *amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

`class` **AmphoraIndexListenersReload**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to reload all listeners on an amphora.

execute(*loadbalancer*, *amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*)

Execute listener reload routines for listeners on an amphora.

`class` **AmphoraIndexUpdateVRRPInterface**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to get and update the VRRP interface device name from amphora.

execute(*amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

`class` **AmphoraIndexVRRPStart**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to start keepalived on an amphora.

This will reload keepalived if it is already running.

execute(*amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`AmphoraIndexVRRPUpdate`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to update the VRRP configuration of an amphora.

`execute`(*loadbalancer_id*, *amphorae_network_config*, *amphora_index*, *amphorae*, *amphorae_status: dict*, *amp_vrrp_int: Optional[str]*, *new_amphora_id: str*, *timeout_dict=None*)

Execute update_vrrp_conf.

`class` **`AmphoraPostNetworkPlug`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to notify the amphora post network plug.

`execute`(*amphora*, *ports*, *amphora_network_config*)

Execute post_network_plug routine.

`revert`(*result*, *amphora*, **args*, ***kwargs*)

Handle a failed post network plug.

`class` **`AmphoraPostVIPPlug`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to notify the amphora post VIP plug.

`execute`(*amphora*, *loadbalancer*, *amphorae_network_config*)

Execute post_vip_routine.

`revert`(*result*, *amphora*, *loadbalancer*, **args*, ***kwargs*)

Handle a failed amphora vip plug notification.

`class` **`AmphoraUpdateVRRPInterface`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to get and update the VRRP interface device name from amphora.

`execute`(*amphora*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

`class` **AmphoraVRRPStart**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to start keepalived on an amphora.

This will reload keepalived if it is already running.

execute(*amphora*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

`class` **AmphoraVRRPUpdate**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to update the VRRP configuration of an amphora.

execute(*loadbalancer_id*, *amphorae_network_config*, *amphora*, *amp_vrrp_int*, *timeout_dict=None*)

Execute update_vrrp_conf.

`class` **AmphoraeGetConnectivityStatus**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task that checks amphorae connectivity status.

Check and return the connectivity status of both amphorae in ACTIVE STANDBY load balancers

execute(*amphorae: List[dict]*, *new_amphora_id: str*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **AmphoraePostNetworkPlug**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to notify the amphorae post network plug.

execute(*loadbalancer*, *updated_ports*, *amphorae_network_config*)

Execute post_network_plug routine.

revert(*result*, *loadbalancer*, *updated_ports*, **args*, ***kwargs*)

Handle a failed post network plug.

class **AmphoraePostVIPPlug**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to notify the amphorae post VIP plug.

execute(*loadbalancer*, *amphorae_network_config*)

Execute post_vip_plug across the amphorae.

class **BaseAmphoraTask**(***kwargs*)

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

class **ListenerDelete**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to delete the listener on the vip.

execute(*listener*)

Execute listener delete routines for an amphora.

revert(*listener*, **args*, ***kwargs*)

Handle a failed listener delete.

class **ListenersStart**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to start all listeners on the vip.

execute(*loadbalancer*, *amphora=None*)
Execute listener start routines for listeners on an amphora.

revert(*loadbalancer*, **args*, ***kwargs*)
Handle failed listeners starts.

`class` **ListenersUpdate**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to update amphora with all specified listeners' configurations.

execute(*loadbalancer*)
Execute updates per listener for an amphora.

revert(*loadbalancer*, **args*, ***kwargs*)
Handle failed listeners updates.

octavia.controller.worker.v1.tasks.cert_task module

`class` **BaseCertTask**(***kwargs*)
Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

`class` **GenerateServerPEMTask**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.cert_task.BaseCertTask*

Create the server certs for the agent comm

Use the amphora_id for the CN

execute(*amphora_id*)
Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

octavia.controller.worker.v1.tasks.compute_tasks module

`class` **`AttachPort`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

`execute`(*amphora*, *port*)

Attach a port to an amphora instance.

Parameters

- **`amphora`** -- The amphora to attach the port to.
- **`port`** -- The port to attach to the amphora.

Returns None

`revert`(*amphora*, *port*, **args*, ***kwargs*)

Revert our port attach.

Parameters

- **`amphora`** -- The amphora to detach the port from.
- **`port`** -- The port to attach to the amphora.

`class` **`BaseComputeTask`**(***kwargs*)

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

`class` **`CertComputeCreate`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.ComputeCreate`

`execute`(*amphora_id*, *server_pem*, *server_group_id*, *build_type_priority=40*, *ports=None*, *flavor=None*, *availability_zone=None*)

Create an amphora

Returns an amphora

`class` **`ComputeActiveWait`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

Wait for the compute driver to mark the amphora active.

`execute`(*compute_id*, *amphora_id*, *availability_zone*)

Wait for the compute driver to mark the amphora active

Raises Generic exception if the amphora is not active

Returns An amphora object

`class` **`ComputeCreate`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

Create the compute instance for a new amphora.

`execute`(*amphora_id*, *server_group_id*, *config_drive_files=None*, *build_type_priority=40*, *ports=None*, *flavor=None*, *availability_zone=None*)

Create an amphora

Returns an amphora

revert(*result*, *amphora_id*, **args*, ***kwargs*)

This method will revert the creation of the

amphora. So it will just delete it in this flow

class ComputeDelete(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

execute(*amphora*, *passive_failure=False*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class DeleteAmphoraeOnLoadBalancer(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

Delete the amphorae on a load balancer.

Iterate through amphorae, deleting them

execute(*loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class NovaServerGroupCreate(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

execute(*loadbalancer_id*)

Create a server group by nova client api

Parameters

- **loadbalancer_id** -- will be used for server group's name

- **`policy`** -- will used for server group's policy

Raises Generic exception if the server group is not created

Returns server group's id

`revert`(*result*, **args*, ***kwargs*)

This method will revert the creation of the

Parameters **`result`** -- here it refers to server group id

`class` **`NovaServerGroupDelete`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.compute_tasks.BaseComputeTask`

`execute`(*server_group_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

octavia.controller.worker.v1.tasks.database_tasks module

`class` **`AssociateFailoverAmphoraWithLBID`**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Associate failover amphora with loadbalancer in the database.

`execute`(*amphora_id*, *loadbalancer_id*)

Associate failover amphora with loadbalancer in the database.

Parameters

- **`amphora_id`** -- Id of an amphora to update
- **`loadbalancer_id`** -- Id of a load balancer to be associated with a given amphora.

Returns None

`revert`(*amphora_id*, **args*, ***kwargs*)

Remove amphora-load balancer association.

Parameters **`amphora_id`** -- Id of an amphora that couldn't be associated with a load balancer.

Returns None

`class BaseDatabaseTask(**kwargs)`

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

`class CountPoolChildrenForQuota(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Counts the pool child resources for quota management.

Since the children of pools are cleaned up by the sqlalchemy cascade delete settings, we need to collect the quota counts for the child objects early.

`execute(pool)`

Count the pool child resources for quota management

Parameters `pool` -- The pool to count children on

Returns None

`class CreateAmphoraInDB(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Task to create an initial amphora in the Database.

`execute(*args, loadbalancer_id=None, **kwargs)`

Creates an pending create amphora record in the database.

Returns The created amphora object

`revert(result, *args, **kwargs)`

Revert by storing the amphora in error state in the DB

In a future version we might change the status to DELETED if deleting the amphora was successful

Parameters `result` -- Id of created amphora.

Returns None

`class CreateVRRPGroupForLB(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Create a VRRP group for a load balancer.

`execute(loadbalancer_id)`

Create a VRRP group for a load balancer.

Parameters `loadbalancer_id` -- Load balancer ID for which a VRRP group should be created

`class DecrementHealthMonitorQuota(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Decrements the health monitor quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute(health_mon)`

Decrements the health monitor quota.

Parameters `health_mon` -- The health monitor to decrement the quota on.

Returns None

revert(*health_mon*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **health_mon** -- The health monitor to decrement the quota on.

Returns None

class DecrementL7policyQuota(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Decrements the l7policy quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Decrements the l7policy quota.

Parameters **l7policy** -- The l7policy to decrement the quota on.

Returns None

revert(*l7policy*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **l7policy** -- The l7policy to decrement the quota on.

Returns None

class DecrementL7ruleQuota(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Decrements the l7rule quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*)

Decrements the l7rule quota.

Parameters **l7rule** -- The l7rule to decrement the quota on.

Returns None

revert(*l7rule*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **l7rule** -- The l7rule to decrement the quota on.

Returns None

class DecrementListenerQuota(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Decrements the listener quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*)

Decrements the listener quota.

Parameters **listener** -- The listener to decrement the quota on.

Returns None

revert(*listener*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **`listener`** -- The listener to decrement the quota on.

Returns None

`class DecrementLoadBalancerQuota(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Decrements the load balancer quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*loadbalancer*)

Decrements the load balancer quota.

Parameters **`loadbalancer`** -- The load balancer to decrement the quota on.

Returns None

`revert`(*loadbalancer*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **`loadbalancer`** -- The load balancer to decrement the quota on.

Returns None

`class DecrementMemberQuota(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Decrements the member quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*member*)

Decrements the member quota.

Parameters **`member`** -- The member to decrement the quota on.

Returns None

`revert`(*member*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **`member`** -- The member to decrement the quota on.

Returns None

`class DecrementPoolQuota(**kwargs)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Decrements the pool quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*pool*, *pool_child_count*)

Decrements the pool quota.

Parameters **`pool`** -- The pool to decrement the quota on

Returns None

`revert`(*pool*, *pool_child_count*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **`project_id`** -- The id of project to decrement the quota on

Returns None

class DeleteHealthMonitorInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Delete the health monitor in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*)

Delete the health monitor in DB

Parameters **health_mon** -- The health monitor which should be deleted

Returns None

revert(*health_mon*, **args*, ***kwargs*)

Mark the health monitor ERROR since the mark active couldn't happen

Parameters **health_mon** -- The health monitor which couldn't be deleted

Returns None

class DeleteHealthMonitorInDBByPool(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.DeleteHealthMonitorInDB*

Delete the health monitor in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*)

Delete the health monitor in the DB.

Parameters **pool** -- A pool which health monitor should be deleted.

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the health monitor ERROR since the mark active couldn't happen

Parameters **pool** -- A pool which health monitor couldn't be deleted

Returns None

class DeleteL7PolicyInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Delete the L7 policy in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Delete the l7policy in DB

Parameters **l7policy** -- The l7policy to be deleted

Returns None

revert(*l7policy*, **args*, ***kwargs*)

Mark the l7policy ERROR since the delete couldn't happen

Parameters **l7policy** -- L7 policy that failed to get deleted

Returns None

class DeleteL7RuleInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Delete the L7 rule in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*)

Delete the l7rule in DB

Parameters **l7rule** -- The l7rule to be deleted

Returns None

revert(*l7rule*, **args*, ***kwargs*)

Mark the l7rule ERROR since the delete couldn't happen

Parameters **l7rule** -- L7 rule that failed to get deleted

Returns None

class DeleteListenerInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Delete the listener in the DB.

execute(*listener*)

Delete the listener in DB

Parameters **listener** -- The listener to delete

Returns None

revert(*listener*, **args*, ***kwargs*)

Mark the listener ERROR since the listener didn't delete

Parameters **listener** -- Listener that failed to get deleted

Returns None

class DeleteMemberInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Delete the member in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*)

Delete the member in the DB

Parameters **member** -- The member to be deleted

Returns None

revert(*member*, **args*, ***kwargs*)

Mark the member ERROR since the delete couldn't happen

Parameters **member** -- Member that failed to get deleted

Returns None

class DeletePoolInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Delete the pool in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*)

Delete the pool in DB

Parameters **pool** -- The pool to be deleted

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the pool ERROR since the delete couldn't happen

Parameters **pool** -- Pool that failed to get deleted

Returns None

class **DisableAmphoraHealthMonitoring**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Disable amphora health monitoring.

This disables amphora health monitoring by removing it from the amphora_health table.

execute(*amphora*)

Disable health monitoring for an amphora

Parameters **amphora** -- The amphora to disable health monitoring for

Returns None

class **DisableLBAmphoraeHealthMonitoring**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Disable health monitoring on the LB amphorae.

This disables amphora health monitoring by removing it from the amphora_health table for each amphora on a load balancer.

execute(*loadbalancer*)

Disable health monitoring for amphora on a load balancer

Parameters **loadbalancer** -- The load balancer to disable health monitoring on

Returns None

class **GetAmphoraDetails**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Task to retrieve amphora network details.

execute(*amphora*)

Retrieve amphora network details.

Parameters **amphora** -- Amphora which network details are required

Returns data_models.Amphora object

class **GetAmphoraeFromLoadbalancer**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Task to pull the amphorae from a loadbalancer.

execute(*loadbalancer_id*)

Pull the amphorae from a loadbalancer.

Parameters **`loadbalancer_id`** -- Load balancer ID to get amphorae from

Returns A list of Listener objects

`class` **`GetListenersFromLoadbalancer`**(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`*

Task to pull the listeners from a loadbalancer.

`execute`(*loadbalancer*)

Pull the listeners from a loadbalancer.

Parameters **`loadbalancer`** -- Load balancer which listeners are required

Returns A list of Listener objects

`class` **`GetLoadBalancer`**(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`*

Get an load balancer object from the database.

`execute`(*loadbalancer_id*, **args*, ***kwargs*)

Get an load balancer object from the database.

Parameters **`loadbalancer_id`** -- The load balancer ID to lookup

Returns The load balancer object

`class` **`GetVipFromLoadbalancer`**(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`*

Task to pull the vip from a loadbalancer.

`execute`(*loadbalancer*)

Pull the vip from a loadbalancer.

Parameters **`loadbalancer`** -- Load balancer which VIP is required

Returns VIP associated with a given load balancer

`class` **`MarkAmphoraAllocatedInDB`**(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`*

Will mark an amphora as allocated to a load balancer in the database.

Assume sqlalchemy made sure the DB got retried sufficiently - so just abort

`execute`(*amphora*, *loadbalancer_id*)

Mark amphora as allocated to a load balancer in DB.

Parameters

- **`amphora`** -- Amphora to be updated.
- **`loadbalancer_id`** -- Id of a load balancer to which an amphora should be allocated.

Returns None

`revert`(*result*, *amphora*, *loadbalancer_id*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters

- **`result`** -- Execute task result

- **`amphora`** -- Amphora that was updated.
- **`loadbalancer_id`** -- Id of a load balancer to which an amphora failed to be allocated.

Returns None

`class` **`MarkAmphoraBackupInDB`**`(`*`**kwargs`*`)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks._MarkAmphoraRoleAndPriorityInDB`

Alter the amphora role to: Backup.

`execute``(`*`amphora`*`)`

Mark amphora as BACKUP in db.

Parameters **`amphora`** -- Amphora to update role.

Returns None

`revert``(`*`result`*, *`amphora`*, *`*args`*, *`**kwargs`*`)`

Removes amphora role association.

Parameters **`amphora`** -- Amphora to update role.

Returns None

`class` **`MarkAmphoraBootingInDB`**`(`*`**kwargs`*`)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the amphora as booting in the database.

`execute``(`*`amphora_id`*, *`compute_id`*`)`

Mark amphora booting in DB.

Parameters

- **`amphora_id`** -- Id of the amphora to update
- **`compute_id`** -- Id of a compute on which an amphora resides

Returns None

`revert``(`*`result`*, *`amphora_id`*, *`compute_id`*, *`*args`*, *`**kwargs`*`)`

Mark the amphora as broken and ready to be cleaned up.

Parameters

- **`result`** -- Execute task result
- **`amphora_id`** -- Id of the amphora that failed to update
- **`compute_id`** -- Id of a compute on which an amphora resides

Returns None

`class` **`MarkAmphoraDeletedInDB`**`(`*`**kwargs`*`)`

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the amphora deleted in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute``(`*`amphora`*`)`

Mark the amphora as deleted in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class **MarkAmphoraHealthBusy**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark amphora health monitoring busy.

This prevents amphora failover by marking the amphora busy in the amphora_health table.

execute(*amphora*)

Mark amphora health monitoring busy

Parameters **amphora** -- The amphora to mark amphora health busy

Returns None

class **MarkAmphoraMasterInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks._MarkAmphoraRoleAndPriorityInDB`

Alter the amphora role to: MASTER.

execute(*amphora*)

Mark amphora as MASTER in db.

Parameters **amphora** -- Amphora to update role.

Returns None

revert(*result*, *amphora*, **args*, ***kwargs*)

Removes amphora role association.

Parameters **amphora** -- Amphora to update role.

Returns None

class **MarkAmphoraPendingDeleteInDB**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the amphora pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*amphora*)

Mark the amphora as pending delete in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class MarkAmphoraPendingUpdateInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the amphora pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*amphora*)

Mark the amphora as pending update in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class MarkAmphoraReadyInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

This task will mark an amphora as ready in the database.

Assume sqlalchemy made sure the DB got retried sufficiently - so just abort

execute(*amphora*)

Mark amphora as ready in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class MarkAmphoraStandAloneInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks._MarkAmphoraRoleAndPriorityInDB`

Alter the amphora role to: Standalone.

execute(*amphora*)

Mark amphora as STANDALONE in db.

Parameters **amphora** -- Amphora to update role.

Returns None

revert(*result*, *amphora*, **args*, ***kwargs*)

Removes amphora role association.

Parameters **amphora** -- Amphora to update role.

Returns None

class MarkHealthMonitorActiveInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*health_mon*)
Mark the health monitor ACTIVE in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

`revert`(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class **MarkHealthMonitorPendingCreateInDB**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*health_mon*)
Mark the health monitor as pending create in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

`revert`(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class **MarkHealthMonitorPendingDeleteInDB**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*health_mon*)
Mark the health monitor as pending delete in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

`revert`(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class **MarkHealthMonitorPendingUpdateInDB**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*health_mon*)

Mark the health monitor as pending update in DB.

Parameters **`health_mon`** -- Health Monitor object to be updated

Returns None

`revert`(*health_mon*, **args*, ***kwargs*)

Mark the health monitor as broken

Parameters **`health_mon`** -- Health Monitor object that failed to update

Returns None

`class MarkHealthMonitorsOnlineInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`*

Mark all enabled health monitors Online

Parameters **`loadbalancer`** -- The Load Balancer that has associated health monitors

Returns None

`execute`(*loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class MarkL7PolicyActiveInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`*

Mark the l7policy ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7policy*)

Mark the l7policy ACTIVE in DB.

Parameters **`l7policy`** -- L7Policy object to be updated

Returns None

`revert`(*l7policy*, **args*, ***kwargs*)

Mark the l7policy as broken

Parameters **`l7policy`** -- L7Policy object that failed to update

Returns None

class **MarkL7PolicyPendingCreateInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7policy pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Mark the l7policy as pending create in DB.

Parameters **l7policy** -- L7Policy object to be updated

Returns None

revert(*l7policy*, **args*, ***kwargs*)

Mark the l7policy as broken

Parameters **l7policy** -- L7Policy object that failed to update

Returns None

class **MarkL7PolicyPendingDeleteInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7policy pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Mark the l7policy as pending delete in DB.

Parameters **l7policy** -- L7Policy object to be updated

Returns None

revert(*l7policy*, **args*, ***kwargs*)

Mark the l7policy as broken

Parameters **l7policy** -- L7Policy object that failed to update

Returns None

class **MarkL7PolicyPendingUpdateInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7policy pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Mark the l7policy as pending update in DB.

Parameters **l7policy** -- L7Policy object to be updated

Returns None

revert(*l7policy*, **args*, ***kwargs*)

Mark the l7policy as broken

Parameters **l7policy** -- L7Policy object that failed to update

Returns None

class MarkL7RuleActiveInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7rule ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*)

Mark the l7rule ACTIVE in DB.

Parameters **l7rule** -- L7Rule object to be updated

Returns None

revert(*l7rule*, **args*, ***kwargs*)

Mark the l7rule as broken

Parameters **l7rule** -- L7Rule object that failed to update

Returns None

class MarkL7RulePendingCreateInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7rule pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*)

Mark the l7rule as pending create in DB.

Parameters **l7rule** -- L7Rule object to be updated

Returns None

revert(*l7rule*, **args*, ***kwargs*)

Mark the l7rule as broken

Parameters **l7rule** -- L7Rule object that failed to update

Returns None

class MarkL7RulePendingDeleteInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7rule pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*)

Mark the l7rule as pending delete in DB.

Parameters **l7rule** -- L7Rule object to be updated

Returns None

revert(*l7rule*, **args*, ***kwargs*)

Mark the l7rule as broken

Parameters **l7rule** -- L7Rule object that failed to update

Returns None

class MarkL7RulePendingUpdateInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the l7rule pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*)

Mark the l7rule as pending update in DB.

Parameters **l7rule** -- L7Rule object to be updated

Returns None

revert(*l7rule*, **args*, ***kwargs*)

Mark the l7rule as broken

Parameters **l7rule** -- L7Rule object that failed to update

Returns None

class MarkLBActiveInDB(*mark_subobjects=False*, ***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the load balancer active in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*)

Mark the load balancer as active in DB.

This also marks ACTIVE all sub-objects of the load balancer if self.mark_subobjects is True.

Parameters **loadbalancer** -- Load balancer object to be updated

Returns None

revert(*loadbalancer*, **args*, ***kwargs*)

Mark the load balancer as broken and ready to be cleaned up.

This also puts all sub-objects of the load balancer to ERROR state if self.mark_subobjects is True

Parameters **loadbalancer** -- Load balancer object that failed to update

Returns None

class MarkLBAmphoraeDeletedInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Task to mark a list of amphora deleted in the Database.

execute(*loadbalancer*)

Update load balancer's amphorae statuses to DELETED in the database.

Parameters **loadbalancer** -- The load balancer which amphorae should be marked DELETED.

Returns None

class MarkLBAmphoraeHealthBusy(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark amphorae health monitoring busy for the LB.

This prevents amphorae failover by marking each amphora of a given load balancer busy in the amphora_health table.

execute(*loadbalancer*)

Marks amphorae health busy for each amphora on a load balancer

Parameters **loadbalancer** -- The load balancer to mark amphorae health busy

Returns None

class **MarkLBAndListenersActiveInDB**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer and specified listeners active in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*, *listeners*)

Mark the load balancer and listeners as active in DB.

Parameters

- **loadbalancer** -- Load balancer object to be updated
- **listeners** -- Listener objects to be updated

Returns None

revert(*loadbalancer*, *listeners*, **args*, ***kwargs*)

Mark the load balancer and listeners as broken.

Parameters

- **loadbalancer** -- Load balancer object that failed to update
- **listeners** -- Listener objects that failed to update

Returns None

class **MarkLBDeletedInDB**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer deleted in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*)

Mark the load balancer as deleted in DB.

Parameters **loadbalancer** -- Load balancer object to be updated

Returns None

class **MarkLBPendingDeleteInDB**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*)

Mark the load balancer as pending delete in DB.

Parameters **loadbalancer** -- Load balancer object to be updated

Returns None

class MarkListenerDeletedInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the listener deleted in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*)

Mark the listener as deleted in DB

Parameters **listener** -- The listener to be marked deleted

Returns None

revert(*listener*, **args*, ***kwargs*)

Mark the listener ERROR since the delete couldn't happen

Parameters **listener** -- The listener that couldn't be updated

Returns None

class MarkListenerPendingDeleteInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the listener pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*)

Mark the listener as pending delete in DB.

Parameters **listener** -- The listener to be updated

Returns None

revert(*listener*, **args*, ***kwargs*)

Mark the listener as broken and ready to be cleaned up.

Parameters **listener** -- The listener that couldn't be updated

Returns None

class MarkMemberActiveInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the member ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*)

Mark the member ACTIVE in DB.

Parameters **member** -- Member object to be updated

Returns None

revert(*member*, **args*, ***kwargs*)

Mark the member as broken

Parameters **member** -- Member object that failed to update

Returns None

class **MarkMemberPendingCreateInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the member pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*)

Mark the member as pending create in DB.

Parameters **member** -- Member object to be updated

Returns None

revert(*member*, **args*, ***kwargs*)

Mark the member as broken

Parameters **member** -- Member object that failed to update

Returns None

class **MarkMemberPendingDeleteInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the member pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*)

Mark the member as pending delete in DB.

Parameters **member** -- Member object to be updated

Returns None

revert(*member*, **args*, ***kwargs*)

Mark the member as broken

Parameters **member** -- Member object that failed to update

Returns None

class **MarkMemberPendingUpdateInDB**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the member pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*)

Mark the member as pending update in DB.

Parameters **member** -- Member object to be updated

Returns None

revert(*member*, **args*, ***kwargs*)

Mark the member as broken

Parameters **member** -- Member object that failed to update

Returns None

class MarkPoolActiveInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the pool ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*)

Mark the pool ACTIVE in DB.

Parameters **pool** -- Pool object to be updated

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool** -- Pool object that failed to update

Returns None

class MarkPoolPendingCreateInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the pool pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*)

Mark the pool as pending create in DB.

Parameters **pool** -- Pool object to be updated

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool** -- Pool object that failed to update

Returns None

class MarkPoolPendingDeleteInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Mark the pool pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*)

Mark the pool as pending delete in DB.

Parameters **pool** -- Pool object to be updated

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool** -- Pool object that failed to update

Returns None

class MarkPoolPendingUpdateInDB(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Mark the pool pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*)

Mark the pool as pending update in DB.

Parameters **pool** -- Pool object to be updated

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool** -- Pool object that failed to update

Returns None

class ReloadAmphora(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Get an amphora object from the database.

execute(*amphora_id*)

Get an amphora object from the database.

Parameters **amphora_id** -- The amphora ID to lookup

Returns The amphora object

class ReloadLoadBalancer(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Get an load balancer object from the database.

execute(*loadbalancer_id*, **args*, ***kwargs*)

Get an load balancer object from the database.

Parameters **loadbalancer_id** -- The load balancer ID to lookup

Returns The load balancer object

class UpdateAmpFailoverDetails(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask`

Update amphora failover details in the database.

execute(*amphora*, *vip*, *base_port*)

Update amphora failover details in the database.

Parameters

- **amphora** -- The amphora to update
- **vip** -- The VIP object associated with this amphora.
- **base_port** -- The base port object associated with the amphora.

Returns None

class UpdateAmphoraCertBusyToFalse(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the amphora cert_busy flag to be false.

execute(*amphora*)

Update the amphora cert_busy flag to be false.

Parameters **amphora** -- Amphora to be updated.

Returns None

class UpdateAmphoraComputeId(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Associate amphora with a compute in DB.

execute(*amphora_id*, *compute_id*)

Associate amphora with a compute in DB.

Parameters

- **amphora_id** -- Id of the amphora to update
- **compute_id** -- Id of a compute on which an amphora resides

Returns None

class UpdateAmphoraDBCertExpiration(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the amphora expiration date with new cert file date.

execute(*amphora_id*, *server_pem*)

Update the amphora expiration date with new cert file date.

Parameters

- **amphora_id** -- Id of the amphora to update
- **server_pem** -- Certificate in PEM format

Returns None

class UpdateAmphoraInfo(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update amphora with compute instance details.

execute(*amphora_id*, *compute_obj*)

Update amphora with compute instance details.

Parameters

- **amphora_id** -- Id of the amphora to update
- **compute_obj** -- Compute on which an amphora resides

Returns Updated amphora object

class UpdateAmphoraVIPData(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update amphorae VIP data.

execute(*amp_data*)
Update amphorae VIP data.

Parameters **amps_data** -- Amphorae update dicts.

Returns None

class **UpdateAmphoraeVIPData**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update amphorae VIP data.

execute(*amps_data*)
Update amphorae VIP data.

Parameters **amps_data** -- Amphorae update dicts.

Returns None

class **UpdateHealthMonInDB**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the health monitor in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*, *update_dict*)
Update the health monitor in the DB

Parameters

- **health_mon** -- The health monitor to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*health_mon*, **args*, ***kwargs*)
Mark the health monitor ERROR since the update couldn't happen

Parameters **health_mon** -- The health monitor that couldn't be updated

Returns None

class **UpdateL7PolicyInDB**(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the L7 policy in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*, *update_dict*)
Update the L7 policy in the DB

Parameters

- **l7policy** -- The L7 policy to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*l7policy*, **args*, ***kwargs*)
Mark the l7policy ERROR since the update couldn't happen

Parameters `l7policy` -- L7 policy that couldn't be updated

Returns None

class `UpdateL7RuleInDB`(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the L7 rule in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7rule*, *update_dict*)

Update the L7 rule in the DB

Parameters

- **l7rule** -- The L7 rule to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*l7rule*, **args*, ***kwargs*)

Mark the L7 rule ERROR since the update couldn't happen

Parameters `l7rule` -- L7 rule that couldn't be updated

Returns None

class `UpdateLBServerGroupInDB`(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the server group id info for load balancer in DB.

execute(*loadbalancer_id*, *server_group_id*)

Update the server group id info for load balancer in DB.

Parameters

- **loadbalancer_id** -- Id of a load balancer to update
- **server_group_id** -- Id of a server group to associate with the load balancer

Returns None

revert(*loadbalancer_id*, *server_group_id*, **args*, ***kwargs*)

Remove server group information from a load balancer in DB.

Parameters

- **loadbalancer_id** -- Id of a load balancer that failed to update
- **server_group_id** -- Id of a server group that couldn't be associated with the load balancer

Returns None

class `UpdateListenerInDB`(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the listener in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*, *update_dict*)
Update the listener in the DB

Parameters

- **listener** -- The listener to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*listener*, **args*, ***kwargs*)
Mark the listener ERROR since the update couldn't happen

Parameters **listener** -- The listener that couldn't be updated

Returns None

class UpdateLoadbalancerInDB(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the loadbalancer in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*, *update_dict*)
Update the loadbalancer in the DB

Parameters

- **loadbalancer** -- The load balancer to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

class UpdateMemberInDB(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the member in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*, *update_dict*)
Update the member in the DB

Parameters

- **member** -- The member to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*member*, **args*, ***kwargs*)
Mark the member ERROR since the update couldn't happen

Parameters **member** -- The member that couldn't be updated

Returns None

class UpdatePoolInDB(***kwargs*)
Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update the pool in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*, *update_dict*)

Update the pool in the DB

Parameters

- **pool** -- The pool to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*pool*, **args*, ***kwargs*)

Mark the pool ERROR since the update couldn't happen

Parameters **pool** -- The pool that couldn't be updated

Returns None

class UpdatePoolMembersOperatingStatusInDB(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Updates the members of a pool operating status.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool*, *operating_status*)

Update the members of a pool operating status in DB.

Parameters

- **pool** -- Pool object to be updated
- **operating_status** -- Operating status to set

Returns None

class UpdateVIPAfterAllocation(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.database_tasks.BaseDatabaseTask*

Update a VIP associated with a given load balancer.

execute(*loadbalancer_id*, *vip*)

Update a VIP associated with a given load balancer.

Parameters

- **loadbalancer_id** -- Id of a load balancer which VIP should be updated.
- **vip** -- data_models.Vip object with update data.

Returns The load balancer object.

octavia.controller.worker.v1.tasks.lifecycle_tasks module

class **AmphoraIDToErrorOnRevertTask**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to checkpoint Amphora lifecycle milestones.

execute(*amphora_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*amphora_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class **AmphoraToErrorOnRevertTask**(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.AmphoraIDToErrorOnRevertTask`

Task to checkpoint Amphora lifecycle milestones.

execute(*amphora*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*amphora*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class BaseLifecycleTask(***kwargs*)

Bases: `taskflow.task.Task`

Base task to instansiate common classes.

class HealthMonitorToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a member to ERROR on revert.

execute(*health_mon*, *listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*health_mon*, *listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.

- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`class L7PolicyToErrorOnRevertTask`(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a l7policy to ERROR on revert.

`execute`(*l7policy*, *listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`revert`(*l7policy*, *listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`class L7RuleToErrorOnRevertTask`(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a l7rule to ERROR on revert.

`execute`(*l7rule*, *listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`revert`(*l7rule*, *listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`class ListenerToErrorOnRevertTask`(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a listener to ERROR on revert.

`execute`(*listener*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`revert`(*listener*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`class ListenersToErrorOnRevertTask`(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set listeners to ERROR on revert.

execute(*listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class LoadBalancerIDToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set the load balancer to ERROR on revert.

execute(*loadbalancer_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*loadbalancer_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class LoadBalancerToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.LoadBalancerIDToErrorOnRevertTask`

Task to set the load balancer to ERROR on revert.

execute(*loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class MemberToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a member to ERROR on revert.

execute(*member*, *listeners*, *loadbalancer*, *pool*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*member*, *listeners*, *loadbalancer*, *pool*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class MembersToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set members to ERROR on revert.

execute(*members*, *listeners*, *loadbalancer*, *pool*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*members*, *listeners*, *loadbalancer*, *pool*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class PoolToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a pool to ERROR on revert.

execute(*pool*, *listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

revert(*pool*, *listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

octavia.controller.worker.v1.tasks.model_tasks module

class DeleteModelObject(*name=None*, *provides=None*, *requires=None*, *auto_extract=True*, *rebind=None*, *inject=None*, *ignore_list=None*, *revert_rebind=None*, *revert_requires=None*)

Bases: `taskflow.task.Task`

Task to delete an object in a model.

execute(*object*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

class UpdateAttributes(*name=None*, *provides=None*, *requires=None*, *auto_extract=True*, *rebind=None*, *inject=None*, *ignore_list=None*, *revert_rebind=None*, *revert_requires=None*)

Bases: `taskflow.task.Task`

Task to update an object for changes.

execute(*object*, *update_dict*)

Update an object and its associated resources.

Note: This relies on the data_model update() methods to handle complex objects with nested objects (LoadBalancer.vip, Pool.session_persistence, etc.)

Parameters

- **`object`** -- The object will be updated.
- **`update_dict`** -- The updates dictionary.

Returns None

octavia.controller.worker.v1.tasks.network_tasks module

class AdminDownPort(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`

execute(*port_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*result*, *port_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class AllocateVIP(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`

Task to allocate a VIP.

execute(*loadbalancer*)

Allocate a vip to the loadbalancer.

revert(*result*, *loadbalancer*, **args*, ***kwargs*)

Handle a failure to allocate vip.

class AllocateVIPforFailover(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.network_tasks.AllocateVIP`

Task to allocate/validate the VIP for a failover flow.

revert(*result*, *loadbalancer*, **args*, ***kwargs*)

Handle a failure to allocate vip.

class ApplyQos(***kwargs*)

Bases: `octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`

Apply Quality of Services to the VIP

execute(*loadbalancer*, *amps_data=None*, *update_dict=None*)
Apply qos policy on the vrrp ports which are related with vip.

revert(*result*, *loadbalancer*, *amps_data=None*, *update_dict=None*, **args*, ***kwargs*)
Handle a failure to apply QoS to VIP

class ApplyQosAmphora(***kwargs*)
Bases: `octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`

Apply Quality of Services to the VIP

execute(*loadbalancer*, *amp_data=None*, *update_dict=None*)
Apply qos policy on the vrrp ports which are related with vip.

revert(*result*, *loadbalancer*, *amp_data=None*, *update_dict=None*, **args*, ***kwargs*)
Handle a failure to apply QoS to VIP

class BaseNetworkTask(***kwargs*)
Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

property network_driver

class CalculateAmphoraDelta(***kwargs*)
Bases: `octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`

default_provides = 'delta'

execute(*loadbalancer*, *amphora*, *availability_zone*)
Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class CalculateDelta(***kwargs*)
Bases: `octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`

Task to calculate the delta between

the nics on the amphora and the ones we need. Returns a list for plumbing them.

default_provides = 'deltas'

execute(*loadbalancer*, *availability_zone*)
Compute which NICs need to be plugged

for the amphora to become operational.

Parameters

- **`loadbalancer`** -- the loadbalancer to calculate deltas for all amphorae
- **`availability_zone`** -- availability zone metadata dict

Returns dict of octavia.network.data_models.Delta keyed off amphora id

`class CreateVIPBasePort`(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to create the VIP base port for an amphora.

`execute`(*vip*, *vip_sg_id*, *amphora_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`revert`(*result*, *vip*, *vip_sg_id*, *amphora_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *`execute()`* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the *`execute()`* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`class DeallocateVIP`(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to deallocate a VIP.

`execute`(*loadbalancer*)

Deallocate a VIP.

`class DeletePort`(***kwargs*)

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to delete a network port.

`execute`(*port_id*, *passive_failure=False*)

Delete the network port.

class FailoverPreparationForAmphora(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to prepare an amphora for failover.

execute(*amphora*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class GetAmphoraNetworkConfigs(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to retrieve amphora network details.

execute(*loadbalancer*, *amphora=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class GetAmphoraNetworkConfigsByID(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to retrieve amphora network details.

execute(*loadbalancer_id*, *amphora_id=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`GetAmphoraeNetworkConfigs`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to retrieve amphorae network details.

`execute``(`*`loadbalancer_id`*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`GetMemberPorts`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

`execute``(`*`loadbalancer`*, *`amphora`*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`GetPlumbedNetworks`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to figure out the NICS on an amphora.

This will likely move into the amphora driver :returns: Array of networks

default_provides = 'nics'

execute(*amphora*)

Get plumbed networks for the amphora.

class **GetSubnetFromVIP**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to plumb a VIP.

execute(*loadbalancer*)

Plumb a vip to an amphora.

class **GetVIPSecurityGroupID**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

execute(*loadbalancer_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **HandleNetworkDelta**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to plug and unplug networks

Plug or unplug networks based on delta

execute(*amphora*, *delta*)

Handle network plugging based off deltas.

revert(*result*, *amphora*, *delta*, **args*, ***kwargs*)

Handle a network plug or unplug failures.

class **HandleNetworkDeltas**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to plug and unplug networks

Loop through the deltas and plug or unplug networks based on delta

execute(*deltas*, *loadbalancer*)

Handle network plugging based off deltas.

revert(*result*, *deltas*, **args*, ***kwargs*)

Handle a network plug or unplug failures.

`class` **`PlugNetworks`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to plug the networks.

This uses the delta to add all missing networks/nics

`execute``(`*amphora*, *delta*`)`

Update the amphora networks for the delta.

`revert``(`*amphora*, *delta*, **args*, ***kwargs*`)`

Handle a failed network plug by removing all nics added.

`class` **`PlugPorts`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to plug neutron ports into a compute instance.

`execute``(`*amphora*, *ports*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`PlugVIP`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to plumb a VIP.

`execute``(`*loadbalancer*`)`

Plumb a vip to an amphora.

`revert``(`*result*, *loadbalancer*, **args*, ***kwargs*`)`

Handle a failure to plumb a vip.

`class` **`PlugVIPAmpphora`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task to plumb a VIP.

`execute``(`*loadbalancer*, *amphora*, *subnet*`)`

Plumb a vip to an amphora.

`revert``(`*result*, *loadbalancer*, *amphora*, *subnet*, **args*, ***kwargs*`)`

Handle a failure to plumb a vip.

`class` **`RetrievePortIDsOnAmphoraExceptLBNetwork`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask`*

Task retrieving all the port ids on an amphora, except lb network.

execute(*amphora*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **UnPlugNetworks**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to unplug the networks

Loop over all nics and unplug them based on delta

execute(*amphora*, *delta*)

Unplug the networks.

class **UnplugVIP**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to unplug the vip.

execute(*loadbalancer*)

Unplug the vip.

class **UpdateVIP**(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to update a VIP.

execute(*loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class UpdateVIPForDelete(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to update a VIP for listener delete flows.

execute(*loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class UpdateVIPSecurityGroup(***kwargs*)

Bases: *octavia.controller.worker.v1.tasks.network_tasks.BaseNetworkTask*

Task to setup SG for LB.

execute(*loadbalancer_id*)

Task to setup SG for LB.

Task is idempotent and safe to retry.

octavia.controller.worker.v1.tasks.retry_tasks module

class SleepingRetryTimesController(*attempts=1*, *name=None*, *provides=None*, *requires=None*, *auto_extract=True*, *rebind=None*, *revert_all=False*, *interval=1*)

Bases: `taskflow.retry.Times`

A retry controller to attempt subflow retries a number of times.

This retry controller overrides the Times on_failure to inject a sleep interval between retries. It also adds a log message when all of the retries are exhausted.

Parameters

- **attempts** (*int*) -- number of attempts to retry the associated subflow before giving up
- **name** -- Meaningful name for this atom, should be something that is distinguishable and understandable for notification, debugging, storing and any other similar purposes.
- **provides** -- A set, string or list of items that this will be providing (or could provide) to others, used to correlate and associate the thing/s this atom produces, if it produces anything at all.

- **requires** -- A set or list of required inputs for this atom's `execute` method.
- **rebind** -- A dict of key/value pairs used to define argument name conversions for inputs to this atom's `execute` method.
- **revert_all** (*bool*) -- when provided this will cause the full flow to revert when the number of attempts that have been tried has been reached (when false, it will only locally revert the associated subflow)
- **interval** (*int*) -- Interval, in seconds, between retry attempts.

on_failure(*history*, **args*, ***kwargs*)

Makes a decision about the future.

This method will typically use information about prior failures (if this historical failure information is not available or was not persisted the provided history will be empty).

Returns a retry constant (one of):

- `RETRY`: when the controlling flow must be reverted and restarted again (for example with new parameters).
- `REVERT`: when this controlling flow must be completely reverted and the parent flow (if any) should make a decision about further flow execution.
- `REVERT_ALL`: when this controlling flow and the parent flow (if any) must be reverted and marked as a `FAILURE`.

revert(*history*, **args*, ***kwargs*)

Reverts this retry.

On revert call all results that had been provided by previous tries and all errors caused during reversion are provided. This method will be called *only* if a subflow must be reverted without the retry (that is to say that the controller has ran out of resolution options and has either given up resolution or has failed to handle a execution failure).

Parameters

- **args** -- positional arguments that the retry required to execute.
- **kwargs** -- any keyword arguments that the retry required to execute.

Module contents

Submodules

octavia.controller.worker.v1.controller_worker module

class ControllerWorker

Bases: *octavia.common.base_taskflow.BaseTaskFlowEngine*

amphora_cert_rotation(*amphora_id*)

Perform cert rotation for an amphora.

Parameters **amphora_id** -- ID for amphora to rotate

Returns None

Raises *AmphoraNotFound* -- The referenced amphora was not found

`batch_update_members`(*old_member_ids*, *new_member_ids*, *updated_members*)

`create_health_monitor`(*health_monitor_id*)

Creates a health monitor.

Parameters **`pool_id`** -- ID of the pool to create a health monitor on

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_l7policy`(*l7policy_id*)

Creates an L7 Policy.

Parameters **`l7policy_id`** -- ID of the l7policy to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_l7rule`(*l7rule_id*)

Creates an L7 Rule.

Parameters **`l7rule_id`** -- ID of the l7rule to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_listener`(*listener_id*)

Creates a listener.

Parameters **`listener_id`** -- ID of the listener to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_load_balancer`(*load_balancer_id*, *flavor=None*, *availability_zone=None*)

Creates a load balancer by allocating Amphorae.

First tries to allocate an existing Amphora in READY state. If none are available it will attempt to build one specifically for this load balancer.

Parameters **`load_balancer_id`** -- ID of the load balancer to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_member`(*member_id*)

Creates a pool member.

Parameters **`member_id`** -- ID of the member to create

Returns None

Raises `NoSuitablePool` -- Unable to find the node pool

`create_pool`(*pool_id*)

Creates a node pool.

Parameters **`pool_id`** -- ID of the pool to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`delete_amphora`(*amphora_id*)

Deletes an existing Amphora.

Parameters `amphora_id` -- ID of the amphora to delete

Returns None

Raises *`AmphoraNotFound`* -- The referenced Amphora was not found

`delete_health_monitor`(*health_monitor_id*)

Deletes a health monitor.

Parameters `pool_id` -- ID of the pool to delete its health monitor

Returns None

Raises `HMNotFound` -- The referenced health monitor was not found

`delete_l7policy`(*l7policy_id*)

Deletes an L7 policy.

Parameters `l7policy_id` -- ID of the l7policy to delete

Returns None

Raises `L7PolicyNotFound` -- The referenced l7policy was not found

`delete_l7rule`(*l7rule_id*)

Deletes an L7 rule.

Parameters `l7rule_id` -- ID of the l7rule to delete

Returns None

Raises `L7RuleNotFound` -- The referenced l7rule was not found

`delete_listener`(*listener_id*)

Deletes a listener.

Parameters `listener_id` -- ID of the listener to delete

Returns None

Raises `ListenerNotFound` -- The referenced listener was not found

`delete_load_balancer`(*load_balancer_id*, *cascade=False*)

Deletes a load balancer by de-allocating Amphorae.

Parameters `load_balancer_id` -- ID of the load balancer to delete

Returns None

Raises `LBNotFound` -- The referenced load balancer was not found

`delete_member`(*member_id*)

Deletes a pool member.

Parameters `member_id` -- ID of the member to delete

Returns None

Raises `MemberNotFound` -- The referenced member was not found

`delete_pool`(*pool_id*)

Deletes a node pool.

Parameters **`pool_id`** -- ID of the pool to delete

Returns None

Raises **`PoolNotFound`** -- The referenced pool was not found

`failover_amphora`(*amphora_id*, *reraise=False*)

Perform failover operations for an amphora.

Note: This expects the load balancer to already be in provisioning_status=PENDING_UPDATE state.

Parameters

- **`amphora_id`** -- ID for amphora to failover
- **`reraise`** -- If enabled reraise any caught exception

Returns None

Raises *`octavia.common.exceptions.NotFound`* -- The referenced amphora was not found

`failover_loadbalancer`(*load_balancer_id*)

Perform failover operations for a load balancer.

Note: This expects the load balancer to already be in provisioning_status=PENDING_UPDATE state.

Parameters **`load_balancer_id`** -- ID for load balancer to failover

Returns None

Raises **`octavia.commom.exceptions.NotFound`** -- The load balancer was not found.

`update_amphora_agent_config`(*amphora_id*)

Update the amphora agent configuration.

Note: This will update the amphora agent configuration file and update the running configuration for mutatable configuration items.

Parameters **`amphora_id`** -- ID of the amphora to update.

Returns None

`update_health_monitor`(*health_monitor_id*, *health_monitor_updates*)

Updates a health monitor.

Parameters

- **`pool_id`** -- ID of the pool to have it's health monitor updated
- **`health_monitor_updates`** -- Dict containing updated health monitor

Returns None

Raises **`HMNotFound`** -- The referenced health monitor was not found

update_l7policy(*l7policy_id*, *l7policy_updates*)

Updates an L7 policy.

Parameters

- `l7policy_id` -- ID of the l7policy to update
- `l7policy_updates` -- Dict containing updated l7policy attributes

Returns None

Raises `L7PolicyNotFound` -- The referenced l7policy was not found

update_l7rule(*l7rule_id*, *l7rule_updates*)

Updates an L7 rule.

Parameters

- `l7rule_id` -- ID of the l7rule to update
- `l7rule_updates` -- Dict containing updated l7rule attributes

Returns None

Raises `L7RuleNotFound` -- The referenced l7rule was not found

update_listener(*listener_id*, *listener_updates*)

Updates a listener.

Parameters

- `listener_id` -- ID of the listener to update
- `listener_updates` -- Dict containing updated listener attributes

Returns None

Raises `ListenerNotFound` -- The referenced listener was not found

update_load_balancer(*load_balancer_id*, *load_balancer_updates*)

Updates a load balancer.

Parameters

- `load_balancer_id` -- ID of the load balancer to update
- `load_balancer_updates` -- Dict containing updated load balancer

Returns None

Raises `LBNotFound` -- The referenced load balancer was not found

update_member(*member_id*, *member_updates*)

Updates a pool member.

Parameters

- `member_id` -- ID of the member to update
- `member_updates` -- Dict containing updated member attributes

Returns None

Raises `MemberNotFound` -- The referenced member was not found

`update_pool`(*pool_id*, *pool_updates*)

Updates a node pool.

Parameters

- **`pool_id`** -- ID of the pool to update
- **`pool_updates`** -- Dict containing updated pool attributes

Returns None

Raises **`PoolNotFound`** -- The referenced pool was not found

Module contents

octavia.controller.worker.v2 package

Subpackages

octavia.controller.worker.v2.flows package

Submodules

octavia.controller.worker.v2.flows.amphora_flows module

`class` **`AmphoraFlows`**

Bases: `object`

`cert_rotate_amphora_flow`()

Implement rotation for amphora's cert.

1. Create a new certificate
2. Upload the cert to amphora
3. update the newly created certificate info to amphora
4. update the cert_busy flag to be false after rotation

Returns The flow for updating an amphora

`get_amphora_for_lb_failover_subflow`(*prefix*, *role='STANDALONE'*, *failed_amp_vrrp_port_id=None*, *is_vrrp_ipv6=False*)

Creates a new amphora that will be used in a failover flow.

Requires loadbalancer_id, flavor, vip, vip_sg_id, loadbalancer

Provides amphora_id, amphora

Parameters

- **`prefix`** -- The flow name prefix to use on the flow and tasks.
- **`role`** -- The role this amphora will have in the topology.
- **`failed_amp_vrrp_port_id`** -- The base port ID of the failed amp.

- **`is_vrrp_ipv6`** -- True if the base port IP is IPv6.

Returns A Taskflow sub-flow that will create the amphora.

`get_amphora_for_lb_subflow`(*prefix*, *role*)

Create a new amphora for lb.

`get_create_amphora_flow`()

Creates a flow to create an amphora.

Returns The flow for creating the amphora

`get_delete_amphora_flow`(*amphora*, *retry_attempts=5*, *retry_interval=5*)

Creates a subflow to delete an amphora and it's port.

This flow is idempotent and safe to retry.

Parameters

- **`amphora`** -- An amphora dict object.
- **`retry_attempts`** -- The number of times the flow is retried.
- **`retry_interval`** -- The time to wait, in seconds, between retries.

Returns The subflow for deleting the amphora.

Raises *`AmphoraNotFound`* -- The referenced Amphora was not found.

`get_failover_amphora_flow`(*failed_amphora*, *lb_amp_count*)

Get a Taskflow flow to failover an amphora.

1. Build a replacement amphora.
2. Delete the old amphora.
3. Update the amphorae listener configurations.
4. Update the VRRP configurations if needed.

Parameters

- **`failed_amphora`** -- The amphora dict to failover.
- **`lb_amp_count`** -- The number of amphora on this load balancer.

Returns The flow that will provide the failover.

`get_vrrp_subflow`(*prefix*, *timeout_dict=None*, *create_vrrp_group=True*, *get_amphorae_status=True*)

`update_amphora_config_flow`()

Creates a flow to update the amphora agent configuration.

Returns The flow for updating an amphora

octavia.controller.worker.v2.flows.flow_utils module

cert_rotate_amphora_flow()

get_batch_update_members_flow(*old_members*, *new_members*, *updated_members*)

get_cascade_delete_load_balancer_flow(*lb*, *listeners=()*, *pools=()*)

get_create_all_listeners_flow()

get_create_amphora_flow()

get_create_health_monitor_flow()

get_create_l7policy_flow()

get_create_l7rule_flow()

get_create_listener_flow()

get_create_load_balancer_flow(*topology*, *listeners=None*)

get_create_member_flow()

get_create_pool_flow()

get_delete_amphora_flow(*amphora*, *retry_attempts=None*, *retry_interval=None*)

get_delete_health_monitor_flow()

get_delete_l7policy_flow()

get_delete_l7rule_flow()

get_delete_listener_flow()

get_delete_load_balancer_flow(*lb*)

get_delete_member_flow()

get_delete_pool_flow()

get_failover_LB_flow(*amps*, *lb*)

get_failover_amphora_flow(*amphora_dict*, *lb_amp_count*)

get_listeners_on_lb(*db_lb*)

Get a list of the listeners on a load balancer.

Parameters **db_lb** -- A load balancer database model object.

Returns A list of provider dict format listeners.

get_pools_on_lb(*db_lb*)

Get a list of the pools on a load balancer.

Parameters **db_lb** -- A load balancer database model object.

Returns A list of provider dict format pools.

get_update_health_monitor_flow()

get_update_l7policy_flow()

get_update_l7rule_flow()

get_update_listener_flow()

get_update_load_balancer_flow()

get_update_member_flow()

get_update_pool_flow()

update_amphora_config_flow()

octavia.controller.worker.v2.flows.health_monitor_flows module

class **HealthMonitorFlows**

Bases: object

get_create_health_monitor_flow()

Create a flow to create a health monitor

Returns The flow for creating a health monitor

get_delete_health_monitor_flow()

Create a flow to delete a health monitor

Returns The flow for deleting a health monitor

get_update_health_monitor_flow()

Create a flow to update a health monitor

Returns The flow for updating a health monitor

octavia.controller.worker.v2.flows.l7policy_flows module

class **L7PolicyFlows**

Bases: object

get_create_l7policy_flow()

Create a flow to create an L7 policy

Returns The flow for creating an L7 policy

get_delete_l7policy_flow()

Create a flow to delete an L7 policy

Returns The flow for deleting an L7 policy

get_update_l7policy_flow()

Create a flow to update an L7 policy

Returns The flow for updating an L7 policy

octavia.controller.worker.v2.flows.l7rule_flows module

class L7RuleFlows

Bases: `object`

get_create_l7rule_flow()

Create a flow to create an L7 rule

Returns The flow for creating an L7 rule

get_delete_l7rule_flow()

Create a flow to delete an L7 rule

Returns The flow for deleting an L7 rule

get_update_l7rule_flow()

Create a flow to update an L7 rule

Returns The flow for updating an L7 rule

octavia.controller.worker.v2.flows.listener_flows module

class ListenerFlows

Bases: `object`

get_create_all_listeners_flow()

Create a flow to create all listeners

Returns The flow for creating all listeners

get_create_listener_flow()

Create a flow to create a listener

Returns The flow for creating a listener

get_delete_listener_flow()

Create a flow to delete a listener

Returns The flow for deleting a listener

get_delete_listener_internal_flow(*listener*)

Create a flow to delete a listener and l7policies internally

(will skip deletion on the amp and marking LB active)

Returns The flow for deleting a listener

get_update_listener_flow()

Create a flow to update a listener

Returns The flow for updating a listener

octavia.controller.worker.v2.flows.load_balancer_flows module

class LoadBalancerFlows

Bases: `object`

get_cascade_delete_load_balancer_flow(*lb*, *listeners*, *pools*)

Creates a flow to delete a load balancer.

Returns The flow for deleting a load balancer

get_create_load_balancer_flow(*topology*, *listeners=None*)

Creates a conditional graph flow that allocates a loadbalancer.

Raises *InvalidTopology* -- Invalid topology specified

Returns The graph flow for creating a loadbalancer.

get_delete_load_balancer_flow(*lb*)

Creates a flow to delete a load balancer.

Returns The flow for deleting a load balancer

get_failover_LB_flow(*amps*, *lb*)

Failover a load balancer.

1. Validate the VIP port is correct and present.
2. Build a replacement amphora.
3. Delete the failed amphora.
4. Configure the replacement amphora listeners.
5. Configure VRRP for the listeners.
6. Build the second replacement amphora.
7. Delete the second failed amphora.
8. Delete any extraneous amphora.
9. Configure the listeners on the new amphorae.
10. Configure the VRRP on the new amphorae.
11. Reload the listener configurations to pick up VRRP changes.
12. Mark the load balancer back to ACTIVE.

Returns The flow that will provide the failover.

get_post_lb_amp_association_flow(*prefix*, *topology*)

Reload the loadbalancer and create networking subflows for

created/allocated amphorae. :return: Post amphorae association subflow

get_update_load_balancer_flow()

Creates a flow to update a load balancer.

Returns The flow for update a load balancer

octavia.controller.worker.v2.flows.member_flows module

class MemberFlows

Bases: `object`

get_batch_update_members_flow(*old_members*, *new_members*, *updated_members*)

Create a flow to batch update members

Returns The flow for batch updating members

get_create_member_flow()

Create a flow to create a member

Returns The flow for creating a member

get_delete_member_flow()

Create a flow to delete a member

Returns The flow for deleting a member

get_update_member_flow()

Create a flow to update a member

Returns The flow for updating a member

octavia.controller.worker.v2.flows.pool_flows module

class PoolFlows

Bases: `object`

get_create_pool_flow()

Create a flow to create a pool

Returns The flow for creating a pool

get_delete_pool_flow()

Create a flow to delete a pool

Returns The flow for deleting a pool

get_delete_pool_flow_internal(*pool_id*)

Create a flow to delete a pool, etc.

Returns The flow for deleting a pool

get_update_pool_flow()

Create a flow to update a pool

Returns The flow for updating a pool

Module contents

octavia.controller.worker.v2.tasks package

Submodules

octavia.controller.worker.v2.tasks.amphora_driver_tasks module

`class` **`AmpListenersUpdate`**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to update the listeners on one amphora.

`execute`(*loadbalancer*, *amphora*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`AmpRetry`**(*attempts=1*, *name=None*, *provides=None*, *requires=None*, *auto_extract=True*, *rebind=None*, *revert_all=False*)

Bases: `taskflow.retry.Times`

`on_failure`(*history*, **args*, ***kwargs*)

Makes a decision about the future.

This method will typically use information about prior failures (if this historical failure information is not available or was not persisted the provided history will be empty).

Returns a retry constant (one of):

- `RETRY`: when the controlling flow must be reverted and restarted again (for example with new parameters).
- `REVERT`: when this controlling flow must be completely reverted and the parent flow (if any) should make a decision about further flow execution.
- `REVERT_ALL`: when this controlling flow and the parent flow (if any) must be reverted and marked as a `FAILURE`.

`class` **`AmphoraCertUpload`**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Upload a certificate to the amphora.

execute(*amphora*, *server_pem*)

Execute cert_update_amphora routine.

class AmphoraComputeConnectivityWait(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to wait for the compute instance to be up.

execute(*amphora*, *raise_retry_exception=False*)

Execute get_info routine for an amphora until it responds.

class AmphoraConfigUpdate(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to push a new amphora agent configuration to the amphora.

execute(*amphora*, *flavor*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class AmphoraFinalize(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to finalize the amphora before any listeners are configured.

execute(*amphora*)

Execute finalize_amphora routine.

revert(*result*, *amphora*, **args*, ***kwargs*)

Handle a failed amphora finalize.

class AmphoraGetDiagnostics(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to get diagnostics on the amphora and the loadbalancers.

execute(*amphora*)

Execute get_diagnostic routine for an amphora.

`class` **`AmphoraGetInfo`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to get information on an amphora.

`execute``(`*amphora*`)`

Execute get_info routine for an amphora.

`class` **`AmphoraIndexListenerUpdate`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to update the listeners on one amphora.

`execute``(`*loadbalancer*, *amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=()*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`AmphoraIndexListenersReload`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to reload all listeners on an amphora.

`execute``(`*loadbalancer*, *amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*`)`

Execute listener reload routines for listeners on an amphora.

`class` **`AmphoraIndexUpdateVRRPInterface`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to get and update the VRRP interface device name from amphora.

`execute``(`*amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`AmphoraIndexVRRPStart`**(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to start keepalived on an amphora.

This will reload keepalived if it is already running.

`execute`(*amphora_index*, *amphorae*, *amphorae_status: dict*, *new_amphora_id: str*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`AmphoraIndexVRRPUpdate`**(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to update the VRRP configuration of an amphora.

`execute`(*loadbalancer_id*, *amphorae_network_config*, *amphora_index*, *amphorae*, *amphorae_status: dict*, *amp_vrrp_int: Optional[str]*, *new_amphora_id: str*, *timeout_dict=None*)

Execute update_vrrp_conf.

`class` **`AmphoraPostNetworkPlug`**(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`*

Task to notify the amphora post network plug.

`execute`(*amphora*, *ports*, *amphora_network_config*)

Execute post_network_plug routine.

`revert`(*result*, *amphora*, **args*, ***kwargs*)

Handle a failed post network plug.

class **AmphoraPostVIPPlug**(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to notify the amphora post VIP plug.

execute(*amphora*, *loadbalancer*, *amphorae_network_config*)

Execute post_vip_routine.

revert(*result*, *amphora*, *loadbalancer*, **args*, ***kwargs*)

Handle a failed amphora vip plug notification.

class **AmphoraUpdateVRRPInterface**(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to get and update the VRRP interface device name from amphora.

execute(*amphora*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **AmphoraVRRPStart**(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to start keepalived on an amphora.

This will reload keepalived if it is already running.

execute(*amphora*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.

- **`kwargs`** -- any keyword arguments that atom requires to execute.

class `AmphoraVRRPUpdate`(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to update the VRRP configuration of an amphora.

`execute`(*loadbalancer_id*, *amphorae_network_config*, *amphora*, *amp_vrrp_int*, *timeout_dict=None*)

Execute update_vrrp_conf.

class `AmphoraeGetConnectivityStatus`(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task that checks amphorae connectivity status.

Check and return the connectivity status of both amphorae in ACTIVE STANDBY load balancers

`execute`(*amphorae: List[dict]*, *new_amphora_id: str*, *timeout_dict=None*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

class `AmphoraePostNetworkPlug`(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to notify the amphorae post network plug.

`execute`(*loadbalancer*, *updated_ports*, *amphorae_network_config*)

Execute post_network_plug routine.

`revert`(*result*, *loadbalancer*, *updated_ports*, **args*, ***kwargs*)

Handle a failed post network plug.

class `AmphoraePostVIPPlug`(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask`

Task to notify the amphorae post VIP plug.

`execute`(*loadbalancer*, *amphorae_network_config*)

Execute post_vip_plug across the amphorae.

class BaseAmphoraTask(***kwargs*)

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

class ListenerDelete(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to delete the listener on the vip.

execute(*listener*)

Execute listener delete routines for an amphora.

revert(*listener*, **args*, ***kwargs*)

Handle a failed listener delete.

class ListenersStart(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to start all listeners on the vip.

execute(*loadbalancer*, *amphora=None*)

Execute listener start routines for listeners on an amphora.

revert(*loadbalancer*, **args*, ***kwargs*)

Handle failed listeners starts.

class ListenersUpdate(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.amphora_driver_tasks.BaseAmphoraTask*

Task to update amphora with all specified listeners' configurations.

execute(*loadbalancer_id*)

Execute updates per listener for an amphora.

revert(*loadbalancer_id*, **args*, ***kwargs*)

Handle failed listeners updates.

octavia.controller.worker.v2.tasks.cert_task module

class BaseCertTask(***kwargs*)

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

class GenerateServerPEMTask(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.cert_task.BaseCertTask*

Create the server certs for the agent comm

Use the amphora_id for the CN

execute(*amphora_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

octavia.controller.worker.v2.tasks.compute_tasks module

`class` **`AttachPort`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`*

`execute``(`*amphora*, *port*`)`

Attach a port to an amphora instance.

Parameters

- **`amphora`** -- The amphora to attach the port to.
- **`port`** -- The port to attach to the amphora.

Returns None

`revert``(`*amphora*, *port*, **args*, ***kwargs*`)`

Revert our port attach.

Parameters

- **`amphora`** -- The amphora to detach the port from.
- **`port`** -- The port to attach to the amphora.

`class` **`BaseComputeTask`**`(`*`**kwargs`*`)`

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

`class` **`CertComputeCreate`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.compute_tasks.ComputeCreate`*

`execute``(`*amphora_id*, *server_pem*, *server_group_id*, *build_type_priority=40*, *ports=None*, *flavor=None*, *availability_zone=None*`)`

Create an amphora

Parameters **`availability_zone`** -- availability zone metadata dictionary

Returns an amphora

`class` **`ComputeCreate`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`*

Create the compute instance for a new amphora.

execute(*amphora_id*, *server_group_id*, *config_drive_files=None*, *build_type_priority=40*, *ports=None*, *flavor=None*, *availability_zone=None*)

Create an amphora

Parameters **availability_zone** -- availability zone metadata dictionary

Returns an amphora

revert(*result*, *amphora_id*, **args*, ***kwargs*)

This method will revert the creation of the

amphora. So it will just delete it in this flow

class ComputeDelete(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`

execute(*amphora*, *passive_failure=False*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class ComputeRetry(*attempts=1*, *name=None*, *provides=None*, *requires=None*, *auto_extract=True*, *rebind=None*, *revert_all=False*)

Bases: `taskflow.retry.Times`

on_failure(*history*, **args*, ***kwargs*)

Makes a decision about the future.

This method will typically use information about prior failures (if this historical failure information is not available or was not persisted the provided history will be empty).

Returns a retry constant (one of):

- `RETRY`: when the controlling flow must be reverted and restarted again (for example with new parameters).
- `REVERT`: when this controlling flow must be completely reverted and the parent flow (if any) should make a decision about further flow execution.
- `REVERT_ALL`: when this controlling flow and the parent flow (if any) must be reverted and marked as a `FAILURE`.

class ComputeWait(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`

Wait for the compute driver to mark the amphora active.

execute(*compute_id*, *amphora_id*, *availability_zone*)
Wait for the compute driver to mark the amphora active

Parameters

- **compute_id** -- virtual machine UUID
- **amphora_id** -- id of the amphora object
- **availability_zone** -- availability zone metadata dictionary

Raises Generic exception if the amphora is not active

Returns An amphora object

class DeleteAmphoraeOnLoadBalancer(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`

Delete the amphorae on a load balancer.

Iterate through amphorae, deleting them

execute(*loadbalancer*)
Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class NovaServerGroupCreate(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`

execute(*loadbalancer_id*)
Create a server group by nova client api

Parameters

- **loadbalancer_id** -- will be used for server group's name
- **policy** -- will used for server group's policy

Raises Generic exception if the server group is not created

Returns server group's id

revert(*result*, **args*, ***kwargs*)
This method will revert the creation of the

Parameters **result** -- here it refers to server group id

class NovaServerGroupDelete(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.compute_tasks.BaseComputeTask`

execute(*server_group_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

octavia.controller.worker.v2.tasks.database_tasks module

`class` **AssociateFailoverAmphoraWithLBID**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Associate failover amphora with loadbalancer in the database.

execute(*amphora_id*, *loadbalancer_id*)

Associate failover amphora with loadbalancer in the database.

Parameters

- **amphora_id** -- Id of an amphora to update
- **loadbalancer_id** -- Id of a load balancer to be associated with a given amphora.

Returns None

revert(*amphora_id*, **args*, ***kwargs*)

Remove amphora-load balancer association.

Parameters **amphora_id** -- Id of an amphora that couldn't be associated with a load balancer.

Returns None

`class` **BaseDatabaseTask**(***kwargs*)

Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

`class` **CountPoolChildrenForQuota**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Counts the pool child resources for quota management.

Since the children of pools are cleaned up by the sqlalchemy cascade delete settings, we need to collect the quota counts for the child objects early.

`execute`(*pool_id*)

Count the pool child resources for quota management

Parameters **`pool_id`** -- pool_id of pool object to count children on

Returns None

`class CreateAmphoraInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Task to create an initial amphora in the Database.

`execute`(**args*, *loadbalancer_id=None*, ***kwargs*)

Creates an pending create amphora record in the database.

Returns The created amphora object

`revert`(*result*, **args*, ***kwargs*)

Revert by storing the amphora in error state in the DB

In a future version we might change the status to DELETED if deleting the amphora was successful

Parameters **`result`** -- Id of created amphora.

Returns None

`class CreateVRRPGroupForLB`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Create a VRRP group for a load balancer.

`execute`(*loadbalancer_id*)

Create a VRRP group for a load balancer.

Parameters **`loadbalancer_id`** -- Load balancer ID for which a VRRP group should be created

`class DecrementHealthMonitorQuota`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Decrements the health monitor quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*project_id*)

Decrements the health monitor quota.

Parameters **`project_id`** -- The project_id to decrement the quota on.

Returns None

`revert`(*project_id*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **`project_id`** -- The project_id to decrement the quota on.

Returns None

`class DecrementL7policyQuota`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Decrements the l7policy quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7policy*)

Decrements the l7policy quota.

Parameters `l7policy` -- The l7policy to decrement the quota on.

Returns None

`revert`(*l7policy*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters `l7policy` -- The l7policy to decrement the quota on.

Returns None

`class DecrementL7ruleQuota`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Decrements the l7rule quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*)

Decrements the l7rule quota.

Parameters `l7rule` -- The l7rule to decrement the quota on.

Returns None

`revert`(*l7rule*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters `l7rule` -- The l7rule to decrement the quota on.

Returns None

`class DecrementListenerQuota`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Decrements the listener quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*project_id*)

Decrements the listener quota.

Parameters `project_id` -- The project_id to decrement the quota on.

Returns None

`revert`(*project_id*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters `project_id` -- The project_id to decrement the quota on.

Returns None

`class DecrementLoadBalancerQuota`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Decrements the load balancer quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*project_id*)

Decrements the load balancer quota.

Parameters **project_id** -- Project id where quota should be reduced

Returns None

revert(*project_id*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **project_id** -- The project id to decrement the quota on.

Returns None

class DecrementMemberQuota(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Decrements the member quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*project_id*)

Decrements the member quota.

Parameters **member** -- The member to decrement the quota on.

Returns None

revert(*project_id*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **member** -- The member to decrement the quota on.

Returns None

class DecrementPoolQuota(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Decrements the pool quota for a project.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*project_id*, *pool_child_count*)

Decrements the pool quota.

Parameters **project_id** -- project_id where the pool to decrement the quota on

Returns None

revert(*project_id*, *pool_child_count*, *result*, **args*, ***kwargs*)

Re-apply the quota

Parameters **project_id** -- The id of project to decrement the quota on

Returns None

class DeleteHealthMonitorInDB(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Delete the health monitor in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*)

Delete the health monitor in DB

Parameters **`health_mon`** -- The health monitor which should be deleted

Returns None

`revert`(*health_mon*, **args*, ***kwargs*)

Mark the health monitor ERROR since the mark active couldn't happen

Parameters **`health_mon`** -- The health monitor which couldn't be deleted

Returns None

class `DeleteHealthMonitorInDBByPool`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.DeleteHealthMonitorInDB`*

Delete the health monitor in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*pool_id*)

Delete the health monitor in the DB.

Parameters **`pool_id`** -- ID of pool which health monitor should be deleted.

Returns None

`revert`(*pool_id*, **args*, ***kwargs*)

Mark the health monitor ERROR since the mark active couldn't happen

Parameters **`pool_id`** -- ID of pool which health monitor couldn't be deleted

Returns None

class `DeleteL7PolicyInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Delete the L7 policy in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7policy*)

Delete the l7policy in DB

Parameters **`l7policy`** -- The l7policy to be deleted

Returns None

`revert`(*l7policy*, **args*, ***kwargs*)

Mark the l7policy ERROR since the delete couldn't happen

Parameters **`l7policy`** -- L7 policy that failed to get deleted

Returns None

class `DeleteL7RuleInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Delete the L7 rule in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*)

Delete the l7rule in DB

Parameters **`l7rule`** -- The l7rule to be deleted

Returns None

`revert`(*l7rule*, **args*, ***kwargs*)
Mark the l7rule ERROR since the delete couldn't happen

Parameters **`l7rule`** -- L7 rule that failed to get deleted

Returns None

`class DeleteListenerInDB`(***kwargs*)
Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Delete the listener in the DB.

`execute`(*listener*)
Delete the listener in DB

Parameters **`listener`** -- The listener to delete

Returns None

`revert`(*listener*, **args*, ***kwargs*)
Mark the listener ERROR since the listener didn't delete

Parameters **`listener`** -- Listener that failed to get deleted

Returns None

`class DeleteMemberInDB`(***kwargs*)
Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Delete the member in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*member*)
Delete the member in the DB

Parameters **`member`** -- The member to be deleted

Returns None

`revert`(*member*, **args*, ***kwargs*)
Mark the member ERROR since the delete couldn't happen

Parameters **`member`** -- Member that failed to get deleted

Returns None

`class DeletePoolInDB`(***kwargs*)
Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Delete the pool in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*pool_id*)
Delete the pool in DB

Parameters **`pool_id`** -- The pool_id to be deleted

Returns None

`revert`(*pool_id*, **args*, ***kwargs*)
Mark the pool ERROR since the delete couldn't happen

Parameters **`pool_id`** -- pool_id that failed to get deleted

Returns None

`class` **`DisableAmphoraHealthMonitoring`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Disable amphora health monitoring.

This disables amphora health monitoring by removing it from the amphora_health table.

`execute``(`*`amphora`*`)`

Disable health monitoring for an amphora

Parameters **`amphora`** -- The amphora to disable health monitoring for

Returns None

`class` **`DisableLBAmphoraeHealthMonitoring`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Disable health monitoring on the LB amphorae.

This disables amphora health monitoring by removing it from the amphora_health table for each amphora on a load balancer.

`execute``(`*`loadbalancer`*`)`

Disable health monitoring for amphora on a load balancer

Parameters **`loadbalancer`** -- The load balancer to disable health monitoring on

Returns None

`class` **`GetAmphoraDetails`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Task to retrieve amphora network details.

`execute``(`*`amphora`*`)`

Retrieve amphora network details.

Parameters **`amphora`** -- Amphora which network details are required

Returns Amphora data dict

`class` **`GetAmphoraeFromLoadbalancer`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Task to pull the amphorae from a loadbalancer.

`execute``(`*`loadbalancer_id`*`)`

Pull the amphorae from a loadbalancer.

Parameters **`loadbalancer_id`** -- Load balancer ID to get amphorae from

Returns A list of Listener objects

`class` **`GetListenersFromLoadbalancer`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Task to pull the listeners from a loadbalancer.

`execute``(`*`loadbalancer`*`)`

Pull the listeners from a loadbalancer.

Parameters **`loadbalancer`** -- Load balancer which listeners are required

Returns A list of Listener objects

class GetLoadBalancer(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Get an load balancer object from the database.

execute(*loadbalancer_id*, **args*, ***kwargs*)

Get an load balancer object from the database.

Parameters **`loadbalancer_id`** -- The load balancer ID to lookup

Returns The load balancer object

class GetVipFromLoadbalancer(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Task to pull the vip from a loadbalancer.

execute(*loadbalancer*)

Pull the vip from a loadbalancer.

Parameters **`loadbalancer`** -- Load balancer which VIP is required

Returns VIP associated with a given load balancer

class MarkAmphoraAllocatedInDB(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Will mark an amphora as allocated to a load balancer in the database.

Assume sqlalchemy made sure the DB got retried sufficiently - so just abort

execute(*amphora*, *loadbalancer_id*)

Mark amphora as allocated to a load balancer in DB.

Parameters

- **`amphora`** -- Amphora to be updated.
- **`loadbalancer_id`** -- Id of a load balancer to which an amphora should be allocated.

Returns None

revert(*result*, *amphora*, *loadbalancer_id*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters

- **`result`** -- Execute task result
- **`amphora`** -- Amphora that was updated.
- **`loadbalancer_id`** -- Id of a load balancer to which an amphora failed to be allocated.

Returns None

class MarkAmphoraBackupInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks._MarkAmphoraRoleAndPriorityInDB`

Alter the amphora role to: Backup.

`execute`(*amphora*)

Mark amphora as BACKUP in db.

Parameters **`amphora`** -- Amphora to update role.

Returns None

`revert`(*result*, *amphora*, **args*, ***kwargs*)

Removes amphora role association.

Parameters **`amphora`** -- Amphora to update role.

Returns None

`class MarkAmphoraBootingInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Mark the amphora as booting in the database.

`execute`(*amphora_id*, *compute_id*)

Mark amphora booting in DB.

Parameters

- **`amphora_id`** -- Id of the amphora to update
- **`compute_id`** -- Id of a compute on which an amphora resides

Returns None

`revert`(*result*, *amphora_id*, *compute_id*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters

- **`result`** -- Execute task result
- **`amphora_id`** -- Id of the amphora that failed to update
- **`compute_id`** -- Id of a compute on which an amphora resides

Returns None

`class MarkAmphoraDeletedInDB`(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Mark the amphora deleted in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*amphora*)

Mark the amphora as deleted in DB.

Parameters **`amphora`** -- Amphora to be updated.

Returns None

`revert`(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **`amphora`** -- Amphora that was updated.

Returns None

class **MarkAmphoraHealthBusy**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark amphora health monitoring busy.

This prevents amphora failover by marking the amphora busy in the amphora_health table.

execute(*amphora*)

Mark amphora health monitoring busy

Parameters **amphora** -- The amphora to mark amphora health busy

Returns None

class **MarkAmphoraMasterInDB**(***kwargs*)

Bases: octavia.controller.worker.v2.tasks.database_tasks._MarkAmphoraRoleAndPriorityInDB

Alter the amphora role to: MASTER.

execute(*amphora*)

Mark amphora as MASTER in db.

Parameters **amphora** -- Amphora to update role.

Returns None

revert(*result*, *amphora*, **args*, ***kwargs*)

Removes amphora role association.

Parameters **amphora** -- Amphora to update role.

Returns None

class **MarkAmphoraPendingDeleteInDB**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the amphora pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*amphora*)

Mark the amphora as pending delete in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class **MarkAmphoraPendingUpdateInDB**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the amphora pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*amphora*)

Mark the amphora as pending update in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class **MarkAmphoraReadyInDB**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

This task will mark an amphora as ready in the database.

Assume sqlalchemy made sure the DB got retried sufficiently - so just abort

execute(*amphora*)

Mark amphora as ready in DB.

Parameters **amphora** -- Amphora to be updated.

Returns None

revert(*amphora*, **args*, ***kwargs*)

Mark the amphora as broken and ready to be cleaned up.

Parameters **amphora** -- Amphora that was updated.

Returns None

class **MarkAmphoraStandAloneInDB**(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks._MarkAmphoraRoleAndPriorityInDB`

Alter the amphora role to: Standalone.

execute(*amphora*)

Mark amphora as STANDALONE in db.

Parameters **amphora** -- Amphora to update role.

Returns None

revert(*result*, *amphora*, **args*, ***kwargs*)

Removes amphora role association.

Parameters **amphora** -- Amphora to update role.

Returns None

class **MarkHealthMonitorActiveInDB**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*)

Mark the health monitor ACTIVE in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

revert(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class MarkHealthMonitorPendingCreateInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*)
Mark the health monitor as pending create in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

revert(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class MarkHealthMonitorPendingDeleteInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*)
Mark the health monitor as pending delete in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

revert(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class MarkHealthMonitorPendingUpdateInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the health monitor pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*)
Mark the health monitor as pending update in DB.

Parameters **health_mon** -- Health Monitor object to be updated

Returns None

revert(*health_mon*, **args*, ***kwargs*)
Mark the health monitor as broken

Parameters **health_mon** -- Health Monitor object that failed to update

Returns None

class MarkHealthMonitorsOnlineInDB(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark all enabled health monitors Online

Parameters **loadbalancer** -- Dictionary of a Load Balancer that has associated health monitors

Returns None

execute(*loadbalancer: dict*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class MarkL7PolicyActiveInDB(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the l7policy ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Mark the l7policy ACTIVE in DB.

Parameters **l7policy** -- L7Policy object to be updated

Returns None

revert(*l7policy*, **args*, ***kwargs*)

Mark the l7policy as broken

Parameters **l7policy** -- L7Policy object that failed to update

Returns None

class MarkL7PolicyPendingCreateInDB(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the l7policy pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*)

Mark the l7policy as pending create in DB.

Parameters **`l7policy`** -- L7Policy object to be updated

Returns None

`revert`(*l7policy*, **args*, ***kwargs*)
Mark the l7policy as broken

Parameters **`l7policy`** -- L7Policy object that failed to update

Returns None

`class` **`MarkL7PolicyPendingDeleteInDB`**(***kwargs*)
Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Mark the l7policy pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7policy*)
Mark the l7policy as pending delete in DB.

Parameters **`l7policy`** -- L7Policy object to be updated

Returns None

`revert`(*l7policy*, **args*, ***kwargs*)
Mark the l7policy as broken

Parameters **`l7policy`** -- L7Policy object that failed to update

Returns None

`class` **`MarkL7PolicyPendingUpdateInDB`**(***kwargs*)
Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Mark the l7policy pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7policy*)
Mark the l7policy as pending update in DB.

Parameters **`l7policy`** -- L7Policy object to be updated

Returns None

`revert`(*l7policy*, **args*, ***kwargs*)
Mark the l7policy as broken

Parameters **`l7policy`** -- L7Policy object that failed to update

Returns None

`class` **`MarkL7RuleActiveInDB`**(***kwargs*)
Bases: *`octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`*

Mark the l7rule ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*)
Mark the l7rule ACTIVE in DB.

Parameters **`l7rule`** -- L7Rule object to be updated

Returns None

`revert`(*l7rule*, **args*, ***kwargs*)
Mark the l7rule as broken

Parameters `l7rule` -- L7Rule object that failed to update

Returns None

class `MarkL7RulePendingCreateInDB`(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the l7rule pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*)
Mark the l7rule as pending create in DB.

Parameters `l7rule` -- L7Rule object to be updated

Returns None

`revert`(*l7rule*, **args*, ***kwargs*)
Mark the l7rule as broken

Parameters `l7rule` -- L7Rule object that failed to update

Returns None

class `MarkL7RulePendingDeleteInDB`(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the l7rule pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*)
Mark the l7rule as pending delete in DB.

Parameters `l7rule` -- L7Rule object to be updated

Returns None

`revert`(*l7rule*, **args*, ***kwargs*)
Mark the l7rule as broken

Parameters `l7rule` -- L7Rule object that failed to update

Returns None

class `MarkL7RulePendingUpdateInDB`(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the l7rule pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*)
Mark the l7rule as pending update in DB.

Parameters `l7rule` -- L7Rule object to be updated

Returns None

revert(*l7rule*, **args*, ***kwargs*)
Mark the l7rule as broken

Parameters **l7rule** -- L7Rule object that failed to update

Returns None

class MarkLBActiveInDB(*mark_subobjects=False*, ***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer active in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*)
Mark the load balancer as active in DB.

This also marks ACTIVE all sub-objects of the load balancer if self.mark_subobjects is True.

Parameters **loadbalancer** -- Load balancer object to be updated

Returns None

revert(*loadbalancer*, **args*, ***kwargs*)
Mark the load balancer as broken and ready to be cleaned up.

This also puts all sub-objects of the load balancer to ERROR state if self.mark_subobjects is True

Parameters **loadbalancer** -- Load balancer object that failed to update

Returns None

class MarkLBActiveInDBByListener(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer active in the DB using a listener dict.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*)
Mark the load balancer as active in DB.

Parameters **listener** -- Listener dictionary

Returns None

class MarkLBAmphoraeDeletedInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Task to mark a list of amphora deleted in the Database.

execute(*loadbalancer*)
Update load balancer's amphorae statuses to DELETED in the database.

Parameters **loadbalancer** -- The load balancer which amphorae should be marked DELETED.

Returns None

class MarkLBAmphoraeHealthBusy(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark amphorae health monitoring busy for the LB.

This prevents amphorae failover by marking each amphora of a given load balancer busy in the amphora_health table.

`execute`(*loadbalancer*)

Marks amphorae health busy for each amphora on a load balancer

Parameters `loadbalancer` -- The load balancer to mark amphorae health busy

Returns None

`class MarkLBAndListenersActiveInDB`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer and specified listeners active in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*loadbalancer_id*, *listeners*)

Mark the load balancer and listeners as active in DB.

Parameters

- `loadbalancer_id` -- The load balancer ID to be updated
- `listeners` -- Listener objects to be updated

Returns None

`revert`(*loadbalancer_id*, *listeners*, **args*, ***kwargs*)

Mark the load balancer and listeners as broken.

Parameters

- `loadbalancer_id` -- The load balancer ID to be updated
- `listeners` -- Listener objects that failed to update

Returns None

`class MarkLBDeletedInDB`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer deleted in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*loadbalancer*)

Mark the load balancer as deleted in DB.

Parameters `loadbalancer` -- Load balancer object to be updated

Returns None

`class MarkLBPendingDeleteInDB`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the load balancer pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*loadbalancer*)

Mark the load balancer as pending delete in DB.

Parameters `loadbalancer` -- Load balancer object to be updated

Returns None

class MarkListenerDeletedInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the listener deleted in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*)

Mark the listener as deleted in DB

Parameters **listener** -- The listener to be marked deleted

Returns None

revert(*listener*, **args*, ***kwargs*)

Mark the listener ERROR since the delete couldn't happen

Parameters **listener** -- The listener that couldn't be updated

Returns None

class MarkListenerPendingDeleteInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the listener pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*)

Mark the listener as pending delete in DB.

Parameters **listener** -- The listener to be updated

Returns None

revert(*listener*, **args*, ***kwargs*)

Mark the listener as broken and ready to be cleaned up.

Parameters **listener** -- The listener that couldn't be updated

Returns None

class MarkMemberActiveInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the member ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*)

Mark the member ACTIVE in DB.

Parameters **member** -- Member object to be updated

Returns None

revert(*member*, **args*, ***kwargs*)

Mark the member as broken

Parameters **member** -- Member object that failed to update

Returns None

`class` **`MarkMemberPendingCreateInDB`**`(**kwargs)`

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the member pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute``(member)`

Mark the member as pending create in DB.

Parameters **`member`** -- Member object to be updated

Returns None

`revert``(member, *args, **kwargs)`

Mark the member as broken

Parameters **`member`** -- Member object that failed to update

Returns None

`class` **`MarkMemberPendingDeleteInDB`**`(**kwargs)`

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the member pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute``(member)`

Mark the member as pending delete in DB.

Parameters **`member`** -- Member object to be updated

Returns None

`revert``(member, *args, **kwargs)`

Mark the member as broken

Parameters **`member`** -- Member object that failed to update

Returns None

`class` **`MarkMemberPendingUpdateInDB`**`(**kwargs)`

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the member pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute``(member)`

Mark the member as pending update in DB.

Parameters **`member`** -- Member object to be updated

Returns None

`revert``(member, *args, **kwargs)`

Mark the member as broken

Parameters **`member`** -- Member object that failed to update

Returns None

class MarkPoolActiveInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the pool ACTIVE in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool_id*)

Mark the pool ACTIVE in DB.

Parameters **pool_id** -- pool_id to be updated

Returns None

revert(*pool_id*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool_id** -- pool_id that failed to update

Returns None

class MarkPoolPendingCreateInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the pool pending create in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool_id*)

Mark the pool as pending create in DB.

Parameters **pool_id** -- pool_id of pool object to be updated

Returns None

revert(*pool_id*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool_id** -- pool_id of pool object that failed to update

Returns None

class MarkPoolPendingDeleteInDB(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Mark the pool pending delete in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool_id*)

Mark the pool as pending delete in DB.

Parameters **pool_id** -- pool_id of pool object to be updated

Returns None

revert(*pool_id*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool_id** -- pool_id of pool object that failed to update

Returns None

`class MarkPoolPendingUpdateInDB`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Mark the pool pending update in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool_id*)

Mark the pool as pending update in DB.

Parameters **pool_id** -- pool_id of pool object to be updated

Returns None

revert(*pool_id*, **args*, ***kwargs*)

Mark the pool as broken

Parameters **pool_id** -- pool_id of pool object that failed to update

Returns None

`class ReloadAmphora`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Get an amphora object from the database.

execute(*amphora*)

Get an amphora object from the database.

Parameters **amphora_id** -- The amphora ID to lookup

Returns The amphora object

`class ReloadLoadBalancer`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Get an load balancer object from the database.

execute(*loadbalancer_id*, **args*, ***kwargs*)

Get an load balancer object from the database.

Parameters **loadbalancer_id** -- The load balancer ID to lookup

Returns The load balancer object

`class UpdateAmpFailoverDetails`(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update amphora failover details in the database.

execute(*amphora*, *vip*, *base_port*)

Update amphora failover details in the database.

Parameters

- **amphora** -- The amphora to update
- **vip** -- The VIP object associated with this amphora.
- **base_port** -- The base port object associated with the amphora.

Returns None

class **UpdateAmphoraCertBusyToFalse**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the amphora cert_busy flag to be false.

execute(*amphora_id*)

Update the amphora cert_busy flag to be false.

Parameters **amphora** -- Amphora to be updated.

Returns None

class **UpdateAmphoraComputeId**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Associate amphora with a compute in DB.

execute(*amphora_id*, *compute_id*)

Associate amphora with a compute in DB.

Parameters

- **amphora_id** -- Id of the amphora to update
- **compute_id** -- Id of a compute on which an amphora resides

Returns None

class **UpdateAmphoraDBCertExpiration**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the amphora expiration date with new cert file date.

execute(*amphora_id*, *server_pem*)

Update the amphora expiration date with new cert file date.

Parameters

- **amphora_id** -- Id of the amphora to update
- **server_pem** -- Certificate in PEM format

Returns None

class **UpdateAmphoraInfo**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update amphora with compute instance details.

execute(*amphora_id*, *compute_obj*)

Update amphora with compute instance details.

Parameters

- **amphora_id** -- Id of the amphora to update
- **compute_obj** -- Compute on which an amphora resides

Returns Updated amphora object

class **UpdateAmphoraVIPData**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update amphorae VIP data.

execute(*amp_data*)
Update amphorae VIP data.

Parameters **amps_data** -- Amphorae update dicts.

Returns None

class UpdateAmphoraeVIPData(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update amphorae VIP data.

execute(*amps_data*)
Update amphorae VIP data.

Parameters **amps_data** -- Amphorae update dicts.

Returns None

class UpdateHealthMonInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the health monitor in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*health_mon*, *update_dict*)
Update the health monitor in the DB

Parameters

- **health_mon** -- The health monitor to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*health_mon*, **args*, ***kwargs*)
Mark the health monitor ERROR since the update couldn't happen

Parameters **health_mon** -- The health monitor that couldn't be updated

Returns None

class UpdateL7PolicyInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the L7 policy in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*l7policy*, *update_dict*)
Update the L7 policy in the DB

Parameters

- **l7policy** -- The L7 policy to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*l7policy*, **args*, ***kwargs*)
Mark the l7policy ERROR since the update couldn't happen

Parameters **`l7policy`** -- L7 policy that couldn't be updated

Returns None

class **`UpdateL7RuleInDB`**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the L7 rule in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

`execute`(*l7rule*, *update_dict*)

Update the L7 rule in the DB

Parameters

- **`l7rule`** -- The L7 rule to be updated
- **`update_dict`** -- The dictionary of updates to apply

Returns None

`revert`(*l7rule*, **args*, ***kwargs*)

Mark the L7 rule ERROR since the update couldn't happen

Parameters **`l7rule`** -- L7 rule that couldn't be updated

Returns None

class **`UpdateLBServerGroupInDB`**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the server group id info for load balancer in DB.

`execute`(*loadbalancer_id*, *server_group_id*)

Update the server group id info for load balancer in DB.

Parameters

- **`loadbalancer_id`** -- Id of a load balancer to update
- **`server_group_id`** -- Id of a server group to associate with the load balancer

Returns None

`revert`(*loadbalancer_id*, *server_group_id*, **args*, ***kwargs*)

Remove server group information from a load balancer in DB.

Parameters

- **`loadbalancer_id`** -- Id of a load balancer that failed to update
- **`server_group_id`** -- Id of a server group that couldn't be associated with the load balancer

Returns None

class **`UpdateListenerInDB`**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update the listener in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*listener*, *update_dict*)
Update the listener in the DB

Parameters

- **listener** -- The listener to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*listener*, **args*, ***kwargs*)
Mark the listener ERROR since the update couldn't happen

Parameters **listener** -- The listener that couldn't be updated

Returns None

class UpdateLoadbalancerInDB(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Update the loadbalancer in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*loadbalancer*, *update_dict*)
Update the loadbalancer in the DB

Parameters

- **loadbalancer** -- The load balancer to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

class UpdateMemberInDB(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Update the member in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*member*, *update_dict*)
Update the member in the DB

Parameters

- **member** -- The member to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*member*, **args*, ***kwargs*)
Mark the member ERROR since the update couldn't happen

Parameters **member** -- The member that couldn't be updated

Returns None

class UpdatePoolInDB(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask`

Update the pool in the DB.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool_id*, *update_dict*)
Update the pool in the DB

Parameters

- **pool_id** -- The pool_id to be updated
- **update_dict** -- The dictionary of updates to apply

Returns None

revert(*pool_id*, **args*, ***kwargs*)
Mark the pool ERROR since the update couldn't happen

Parameters **pool_id** -- The pool_id that couldn't be updated

Returns None

class UpdatePoolMembersOperatingStatusInDB(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Updates the members of a pool operating status.

Since sqlalchemy will likely retry by itself always revert if it fails

execute(*pool_id*, *operating_status*)
Update the members of a pool operating status in DB.

Parameters

- **pool_id** -- pool_id of pool object to be updated
- **operating_status** -- Operating status to set

Returns None

class UpdateVIPAfterAllocation(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.database_tasks.BaseDatabaseTask*

Update a VIP associated with a given load balancer.

execute(*loadbalancer_id*, *vip*)
Update a VIP associated with a given load balancer.

Parameters

- **loadbalancer_id** -- Id of a load balancer which VIP should be updated.
- **vip** -- data_models.Vip object with update data.

Returns The load balancer object.

octavia.controller.worker.v2.tasks.lifecycle_tasks module

class AmphoraIDToErrorOnRevertTask(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask*

Task to checkpoint Amphora lifecycle milestones.

execute(*amphora_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*amphora_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class AmphoraToErrorOnRevertTask(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.lifecycle_tasks.AmphoraIDToErrorOnRevertTask*

Task to checkpoint Amphora lifecycle milestones.

execute(*amphora*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*amphora*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class BaseLifecycleTask(***kwargs*)

Bases: `taskflow.task.Task`

Base task to instansiate common classes.

class HealthMonitorToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a member to ERROR on revert.

execute(*health_mon*, *listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*health_mon*, *listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.

- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class L7PolicyToErrorOnRevertTask(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask*

Task to set a l7policy to ERROR on revert.

execute(*l7policy*, *listeners*, *loadbalancer_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*l7policy*, *listeners*, *loadbalancer_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *execute()* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the *execute()* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class L7RuleToErrorOnRevertTask(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask*

Task to set a l7rule to ERROR on revert.

execute(*l7rule*, *l7policy_id*, *listeners*, *loadbalancer_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*l7rule*, *l7policy_id*, *listeners*, *loadbalancer_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class ListenerToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a listener to ERROR on revert.

execute(*listener*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*listener*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class ListenersToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a listener to ERROR on revert.

execute(*listeners*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*listeners*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class LoadBalancerIDToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set the load balancer to ERROR on revert.

execute(*loadbalancer_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*loadbalancer_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class LoadBalancerToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.LoadBalancerIDToErrorOnRevertTask`

Task to set the load balancer to ERROR on revert.

execute(*loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class MemberToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a member to ERROR on revert.

execute(*member*, *listeners*, *loadbalancer*, *pool_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*member*, *listeners*, *loadbalancer*, *pool_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class MembersToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set members to ERROR on revert.

execute(*members*, *listeners*, *loadbalancer*, *pool_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*members*, *listeners*, *loadbalancer*, *pool_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class PoolToErrorOnRevertTask(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.lifecycle_tasks.BaseLifecycleTask`

Task to set a pool to ERROR on revert.

execute(*pool_id*, *listeners*, *loadbalancer*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

revert(*pool_id*, *listeners*, *loadbalancer*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

octavia.controller.worker.v2.tasks.network_tasks module

class AdminDownPort(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

execute(*port_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

revert(*result*, *port_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the `execute()` method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **args** -- positional arguments that the atom required to execute.
- **kwargs** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the `execute()` result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

class AllocateVIP(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task to allocate a VIP.

execute(*loadbalancer*)

Allocate a vip to the loadbalancer.

revert(*result*, *loadbalancer*, **args*, ***kwargs*)

Handle a failure to allocate vip.

class AllocateVIPforFailover(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.AllocateVIP`

Task to allocate/validate the VIP for a failover flow.

revert(*result*, *loadbalancer*, **args*, ***kwargs*)

Handle a failure to allocate vip.

class ApplyQos(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Apply Quality of Services to the VIP

execute(*loadbalancer*, *amps_data=None*, *update_dict=None*)
Apply qos policy on the vrrp ports which are related with vip.

revert(*result*, *loadbalancer*, *amps_data=None*, *update_dict=None*, **args*, ***kwargs*)
Handle a failure to apply QoS to VIP

class ApplyQosAmphora(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Apply Quality of Services to the VIP

execute(*loadbalancer*, *amp_data=None*, *update_dict=None*)
Apply qos policy on the vrrp ports which are related with vip.

revert(*result*, *loadbalancer*, *amp_data=None*, *update_dict=None*, **args*, ***kwargs*)
Handle a failure to apply QoS to VIP

class BaseNetworkTask(***kwargs*)
Bases: `taskflow.task.Task`

Base task to load drivers common to the tasks.

property network_driver

class CalculateAmphoraDelta(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

default_provides = 'delta'

execute(*loadbalancer*, *amphora*, *availability_zone*)
Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class CalculateDelta(***kwargs*)
Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task to calculate the delta between

the nics on the amphora and the ones we need. Returns a list for plumbing them.

default_provides = 'deltas'

execute(*loadbalancer*, *availability_zone*)
Compute which NICs need to be plugged

for the amphora to become operational.

Parameters

- **`loadbalancer`** -- the loadbalancer to calculate deltas for all amphorae
- **`availability_zone`** -- availability zone metadata dict

Returns dict of octavia.network.data_models.Delta keyed off amphora id

`class` **`CreateVIPBasePort`**(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`*

Task to create the VIP base port for an amphora.

`execute`(*vip*, *vip_sg_id*, *amphora_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`revert`(*result*, *vip*, *vip_sg_id*, *amphora_id*, **args*, ***kwargs*)

Revert this atom.

This method should undo any side-effects caused by previous execution of the atom using the result of the *`execute()`* method and information on the failure which triggered reversion of the flow the atom is contained in (if applicable).

Parameters

- **`args`** -- positional arguments that the atom required to execute.
- **`kwargs`** -- any keyword arguments that the atom required to execute; the special key `'result'` will contain the *`execute()`* result (if any) and the `**kwargs` key `'flow_failures'` will contain any failure information.

`class` **`DeallocateVIP`**(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`*

Task to deallocate a VIP.

`execute`(*loadbalancer*)

Deallocate a VIP.

`class` **`DeletePort`**(***kwargs*)

Bases: *`octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`*

Task to delete a network port.

`execute`(*port_id*, *passive_failure=False*)

Delete the network port.

`class FailoverPreparationForAmphora(**kwargs)`

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to prepare an amphora for failover.

`execute(amphora)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

`class GetAmphoraNetworkConfigs(**kwargs)`

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to retrieve amphora network details.

`execute(loadbalancer, amphora=None)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

`class GetAmphoraNetworkConfigsByID(**kwargs)`

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to retrieve amphora network details.

`execute(loadbalancer_id, amphora_id=None)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`GetAmphoraeNetworkConfigs`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`*

Task to retrieve amphorae network details.

`execute``(`*`loadbalancer_id`*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`GetMemberPorts`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`*

`execute``(`*`loadbalancer`*, *`amphora`*`)`

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **`args`** -- positional arguments that atom requires to execute.
- **`kwargs`** -- any keyword arguments that atom requires to execute.

`class` **`GetPlumbedNetworks`**`(`*`**kwargs`*`)`

Bases: *`octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`*

Task to figure out the NICS on an amphora.

This will likely move into the amphora driver :returns: Array of networks

default_provides = 'nics'

execute(*amphora*)
Get plumbed networks for the amphora.

class **GetSubnetFromVIP**(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to plumb a VIP.

execute(*loadbalancer*)
Plumb a vip to an amphora.

class **GetVIPSecurityGroupID**(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

execute(*loadbalancer_id*)
Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **HandleNetworkDelta**(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to plug and unplug networks

Plug or unplug networks based on delta

execute(*amphora*, *delta*)
Handle network plugging based off deltas.

revert(*result*, *amphora*, *delta*, **args*, ***kwargs*)
Handle a network plug or unplug failures.

class **HandleNetworkDeltas**(***kwargs*)
Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to plug and unplug networks

Loop through the deltas and plug or unplug networks based on delta

execute(*deltas*, *loadbalancer*)
Handle network plugging based off deltas.

revert(*result*, *deltas*, **args*, ***kwargs*)
Handle a network plug or unplug failures.

class PlugNetworks(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task to plug the networks.

This uses the delta to add all missing networks/nics

execute(*amphora*, *delta*)

Update the amphora networks for the delta.

revert(*amphora*, *delta*, **args*, ***kwargs*)

Handle a failed network plug by removing all nics added.

class PlugPorts(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task to plug neutron ports into a compute instance.

execute(*amphora*, *ports*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class PlugVIP(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task to plumb a VIP.

execute(*loadbalancer*)

Plumb a vip to an amphora.

revert(*result*, *loadbalancer*, **args*, ***kwargs*)

Handle a failure to plumb a vip.

class PlugVIPAmphora(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task to plumb a VIP.

execute(*loadbalancer*, *amphora*, *subnet*)

Plumb a vip to an amphora.

revert(*result*, *loadbalancer*, *amphora*, *subnet*, **args*, ***kwargs*)

Handle a failure to plumb a vip.

class RetrievePortIDsOnAmphoraExceptLBNetwork(***kwargs*)

Bases: `octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask`

Task retrieving all the port ids on an amphora, except lb network.

execute(*amphora*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class **UnPlugNetworks**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to unplug the networks

Loop over all nics and unplug them based on delta

execute(*amphora*, *delta*)

Unplug the networks.

class **UnplugVIP**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to unplug the vip.

execute(*loadbalancer*)

Unplug the vip.

class **UpdateVIP**(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to update a VIP.

execute(*listeners*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class UpdateVIPForDelete(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to update a VIP for listener delete flows.

execute(*loadbalancer_id*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

class UpdateVIPSecurityGroup(***kwargs*)

Bases: *octavia.controller.worker.v2.tasks.network_tasks.BaseNetworkTask*

Task to setup SG for LB.

execute(*loadbalancer_id*)

Task to setup SG for LB.

octavia.controller.worker.v2.tasks.retry_tasks module

class SleepingRetryTimesController(*attempts=1*, *name=None*, *provides=None*, *requires=None*, *auto_extract=True*, *rebind=None*, *revert_all=False*, *interval=1*)

Bases: `taskflow.retry.Times`

A retry controller to attempt subflow retries a number of times.

This retry controller overrides the Times on_failure to inject a sleep interval between retries. It also adds a log message when all of the retries are exhausted.

Parameters

- **attempts** (*int*) -- number of attempts to retry the associated subflow before giving up
- **name** -- Meaningful name for this atom, should be something that is distinguishable and understandable for notification, debugging, storing and any other similar purposes.
- **provides** -- A set, string or list of items that this will be providing (or could provide) to others, used to correlate and associate the thing/s this atom produces, if it produces anything at all.
- **requires** -- A set or list of required inputs for this atom's `execute` method.

- **`rebind`** -- A dict of key/value pairs used to define argument name conversions for inputs to this atom's `execute` method.
- **`revert_all`** (*`bool`*) -- when provided this will cause the full flow to revert when the number of attempts that have been tried has been reached (when false, it will only locally revert the associated subflow)
- **`interval`** (*`int`*) -- Interval, in seconds, between retry attempts.

`on_failure`(*history*, **args*, ***kwargs*)

Makes a decision about the future.

This method will typically use information about prior failures (if this historical failure information is not available or was not persisted the provided history will be empty).

Returns a retry constant (one of):

- `RETRY`: when the controlling flow must be reverted and restarted again (for example with new parameters).
- `REVERT`: when this controlling flow must be completely reverted and the parent flow (if any) should make a decision about further flow execution.
- `REVERT_ALL`: when this controlling flow and the parent flow (if any) must be reverted and marked as a `FAILURE`.

`revert`(*history*, **args*, ***kwargs*)

Reverts this retry.

On revert call all results that had been provided by previous tries and all errors caused during reversion are provided. This method will be called *only* if a subflow must be reverted without the retry (that is to say that the controller has ran out of resolution options and has either given up resolution or has failed to handle a execution failure).

Parameters

- **`args`** -- positional arguments that the retry required to execute.
- **`kwargs`** -- any keyword arguments that the retry required to execute.

Module contents

Submodules

octavia.controller.worker.v2.controller_worker module

`class ControllerWorker`

Bases: `object`

`amphora_cert_rotation`(*amphora_id*)

Perform cert rotation for an amphora.

Parameters **`amphora_id`** -- ID for amphora to rotate

Returns None

Raises *`AmphoraNotFound`* -- The referenced amphora was not found

`batch_update_members`(*old_members*, *new_members*, *updated_members*) `create_health_monitor`(*health_monitor*)

Creates a health monitor.

Parameters `health_monitor` -- Provider health monitor dict

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_l7policy`(*l7policy*)

Creates an L7 Policy.

Parameters `l7policy` -- Provider dict of the l7policy to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_l7rule`(*l7rule*)

Creates an L7 Rule.

Parameters `l7rule` -- Provider dict l7rule

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_listener`(*listener*)

Creates a listener.

Parameters `listener` -- A listener provider dictionary.

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_load_balancer`(*loadbalancer*, *flavor=None*, *availability_zone=None*)

Creates a load balancer by allocating Amphorae.

First tries to allocate an existing Amphora in READY state. If none are available it will attempt to build one specifically for this load balancer.

Parameters `loadbalancer` -- The dict of load balancer to create

Returns None

Raises `NoResultFound` -- Unable to find the object

`create_member`(*member*)

Creates a pool member.

Parameters `member` -- A member provider dictionary to create

Returns None

Raises `NoSuitablePool` -- Unable to find the node pool

`create_pool`(*pool*)

Creates a node pool.

Parameters `pool` -- Provider pool dict to create

Returns None

Raises `NoResultFound` -- Unable to find the object

delete_amphora(*amphora_id*)
Deletes an existing Amphora.

Parameters **amphora_id** -- ID of the amphora to delete

Returns None

Raises *AmphoraNotFound* -- The referenced Amphora was not found

delete_health_monitor(*health_monitor*)
Deletes a health monitor.

Parameters **health_monitor** -- Provider health monitor dict

Returns None

Raises **HMNotFound** -- The referenced health monitor was not found

delete_l7policy(*l7policy*)
Deletes an L7 policy.

Parameters **l7policy** -- Provider dict of the l7policy to delete

Returns None

Raises **L7PolicyNotFound** -- The referenced l7policy was not found

delete_l7rule(*l7rule*)
Deletes an L7 rule.

Parameters **l7rule** -- Provider dict of the l7rule to delete

Returns None

Raises **L7RuleNotFound** -- The referenced l7rule was not found

delete_listener(*listener*)
Deletes a listener.

Parameters **listener** -- A listener provider dictionary to delete

Returns None

Raises **ListenerNotFound** -- The referenced listener was not found

delete_load_balancer(*load_balancer*, *cascade=False*)
Deletes a load balancer by de-allocating Amphorae.

Parameters **load_balancer** -- Dict of the load balancer to delete

Returns None

Raises **LBNotFound** -- The referenced load balancer was not found

delete_member(*member*)
Deletes a pool member.

Parameters **member** -- A member provider dictionary to delete

Returns None

Raises **MemberNotFound** -- The referenced member was not found

delete_pool(*pool*)
Deletes a node pool.

Parameters `pool` -- Provider pool dict to delete

Returns None

Raises `PoolNotFound` -- The referenced pool was not found

`failover_amphora`(*amphora_id*, *reraise=False*)

Perform failover operations for an amphora.

Note: This expects the load balancer to already be in provisioning_status=PENDING_UPDATE state.

Parameters

- `amphora_id` -- ID for amphora to failover
- `reraise` -- If enabled reraise any caught exception

Returns None

Raises *`octavia.common.exceptions.NotFound`* -- The referenced amphora was not found

`failover_loadbalancer`(*load_balancer_id*)

Perform failover operations for a load balancer.

Note: This expects the load balancer to already be in provisioning_status=PENDING_UPDATE state.

Parameters `load_balancer_id` -- ID for load balancer to failover

Returns None

Raises `octavia.commom.exceptions.NotFound` -- The load balancer was not found.

`run_flow`(*func*, **args*, ***kwargs*)

`property services_controller`

`update_amphora_agent_config`(*amphora_id*)

Update the amphora agent configuration.

Note: This will update the amphora agent configuration file and update the running configuration for mutatable configuration items.

Parameters `amphora_id` -- ID of the amphora to update.

Returns None

`update_health_monitor`(*original_health_monitor*, *health_monitor_updates*)

Updates a health monitor.

Parameters

- `original_health_monitor` -- Provider health monitor dict
- `health_monitor_updates` -- Dict containing updated health monitor

Returns None

Raises `HMNotFound` -- The referenced health monitor was not found `update_l7policy`(*original_l7policy*, *l7policy_updates*)

Updates an L7 policy.

Parameters

- `l7policy` -- Provider dict of the l7policy to update
- `l7policy_updates` -- Dict containing updated l7policy attributes

Returns None

Raises `L7PolicyNotFound` -- The referenced l7policy was not found

`update_l7rule`(*original_l7rule*, *l7rule_updates*)

Updates an L7 rule.

Parameters

- `l7rule` -- Origin dict of the l7rule to update
- `l7rule_updates` -- Dict containing updated l7rule attributes

Returns None

Raises `L7RuleNotFound` -- The referenced l7rule was not found

`update_listener`(*listener*, *listener_updates*)

Updates a listener.

Parameters

- `listener` -- A listener provider dictionary to update
- `listener_updates` -- Dict containing updated listener attributes

Returns None

Raises `ListenerNotFound` -- The referenced listener was not found

`update_load_balancer`(*original_load_balancer*, *load_balancer_updates*)

Updates a load balancer.

Parameters

- `original_load_balancer` -- Dict of the load balancer to update
- `load_balancer_updates` -- Dict containing updated load balancer

Returns None

Raises `LBNotFound` -- The referenced load balancer was not found

`update_member`(*member*, *member_updates*)

Updates a pool member.

Parameters

- `member_id` -- A member provider dictionary to update
- `member_updates` -- Dict containing updated member attributes

Returns None

Raises `MemberNotFound` -- The referenced member was not found

update_pool(*origin_pool*, *pool_updates*)

Updates a node pool.

Parameters

- **origin_pool** -- Provider pool dict to update
- **pool_updates** -- Dict containing updated pool attributes

Returns None

Raises **PoolNotFound** -- The referenced pool was not found

retryMaskFilter(*record*)

octavia.controller.worker.v2.taskflow_jobboard_driver module

class **JobboardTaskFlowDriver**

Bases: `object`

abstract **job_board**(*persistence*)

Setting up jobboard backend based on configuration setting.

Parameters **persistence** -- taskflow persistence backend instance

Returns taskflow jobboard backend instance

class **MysqlPersistenceDriver**

Bases: `object`

get_persistence()

initialize()

class **RedisTaskFlowDriver**(*persistence_driver*)

Bases: `octavia.controller.worker.v2.taskflow_jobboard_driver.JobboardTaskFlowDriver`

job_board(*persistence*)

Setting up jobboard backend based on configuration setting.

Parameters **persistence** -- taskflow persistence backend instance

Returns taskflow jobboard backend instance

class **ZookeeperTaskFlowDriver**(*persistence_driver*)

Bases: `octavia.controller.worker.v2.taskflow_jobboard_driver.JobboardTaskFlowDriver`

job_board(*persistence*)

Setting up jobboard backend based on configuration setting.

Parameters **persistence** -- taskflow persistence backend instance

Returns taskflow jobboard backend instance

Module contents

Submodules

octavia.controller.worker.amphora_rate_limit module

class AmphoraBuildRateLimit

Bases: `object`

add_to_build_request_queue(*amphora_id*, *build_priority*)

has_build_slot()

has_highest_priority(*amphora_id*)

remove_all_from_build_req_queue()

remove_from_build_req_queue(*amphora_id*)

update_build_status_and_available_build_slots(*amphora_id*)

wait_for_build_slot(*amphora_id*)

octavia.controller.worker.task_utils module

Methods common to the controller work tasks.

class TaskUtils(***kwargs*)

Bases: `object`

Class of helper/utility methods used by tasks.

get_current_loadbalancer_from_db(*loadbalancer_id*)

Gets a Loadbalancer from db.

Param loadbalancer_id: Load balancer ID which to get from db

mark_amphora_status_error(*amphora_id*)

Sets an amphora status to ERROR.

NOTE: This should only be called from revert methods.

Parameters **amphora_id** -- Amphora ID to set the status to ERROR

mark_health_mon_prov_status_error(*health_mon_id*)

Sets a health monitor provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters **health_mon_id** -- Health Monitor ID to set prov status to ERROR

mark_l7policy_prov_status_active(*l7policy_id*)

Sets a L7 policy provisioning status to ACTIVE.

NOTE: This should only be called from revert methods.

Parameters **l7policy_id** -- L7 Policy ID to set provisioning status to ACTIVE

mark_l7policy_prov_status_error(*l7policy_id*)

Sets a L7 policy provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters `l7policy_id` -- L7 Policy ID to set provisioning status to ERROR

mark_l7rule_prov_status_error(*l7rule_id*)

Sets a L7 rule provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters `l7rule_id` -- L7 Rule ID to set provisioning status to ERROR

mark_listener_prov_status_active(*listener_id*)

Sets a listener provisioning status to ACTIVE.

NOTE: This should only be called from revert methods.

Parameters `listener_id` -- Listener ID to set provisioning status to ACTIVE

mark_listener_prov_status_error(*listener_id*)

Sets a listener provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters `listener_id` -- Listener ID to set provisioning status to ERROR

mark_loadbalancer_prov_status_active(*loadbalancer_id*)

Sets a load balancer provisioning status to ACTIVE.

NOTE: This should only be called from revert methods.

Parameters `loadbalancer_id` -- Load balancer ID to set provisioning status to ACTIVE

mark_loadbalancer_prov_status_error(*loadbalancer_id*)

Sets a load balancer provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters `loadbalancer_id` -- Load balancer ID to set provisioning status to ERROR

mark_member_prov_status_error(*member_id*)

Sets a member provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters `member_id` -- Member ID to set provisioning status to ERROR

mark_pool_prov_status_active(*pool_id*)

Sets a pool provisioning status to ACTIVE.

NOTE: This should only be called from revert methods.

Parameters `pool_id` -- Pool ID to set provisioning status to ACTIVE

mark_pool_prov_status_error(*pool_id*)

Sets a pool provisioning status to ERROR.

NOTE: This should only be called from revert methods.

Parameters `pool_id` -- Pool ID to set provisioning status to ERROR

status_update_retry()

unmark_amphora_health_busy(*amphora_id*)

Unmark the amphora_health record busy for an amphora.

NOTE: This should only be called from revert methods.

Parameters **amphora_id** -- The amphora id to unmark busy

Module contents

Module contents

octavia.db package

Submodules

octavia.db.api module

get_engine()

get_lock_session()

Context manager for using a locking (not auto-commit) session.

get_session(*expire_on_commit=True*, *autocommit=True*)

Helper method to grab session.

wait_for_connection(*exit_event*)

Helper method to wait for DB connection

octavia.db.base_models module

class IdMixin

Bases: `object`

Id mixin, add to subclasses that have an id.

id = Column(None, String(length=36), table=None, primary_key=True, nullable=False, default=ColumnDefault(<function generate_uuid>))

class LookupTableMixin

Bases: `object`

Mixin to add to classes that are lookup tables.

description = Column(None, String(length=255), table=None)

name = Column(None, String(length=255), table=None, primary_key=True, nullable=False)

class NameMixin

Bases: `object`

Name mixin to add to classes which need a name.

name = Column(None, String(length=255), table=None)

class OctaviaBase
Bases: `oslo_db.sqlalchemy.models.ModelBase`

static apply_filter(*query*, *model*, *filters*)

to_data_model(*_graph_nodes=None*)
Converts to a data model graph.

In order to make the resulting data model graph usable no matter how many internal references are followed, we generate a complete graph of OctaviaBase nodes connected to the object passed to this method.

Parameters **_graph_nodes** -- Used only for internal recursion of this method. Should not be called from the outside. Contains a dictionary of all OctaviaBase type objects in the generated graph

class ProjectMixin
Bases: `object`

Tenant mixin, add to subclasses that have a project.

project_id = Column(None, String(length=36), table=None)

class TagMixin
Bases: `object`

Tags mixin to add to classes which need tags.

The class must realize the specified db relationship as well.

property tags

class Tags(***kwargs*)
Bases: `sqlalchemy.orm.decl_api.Base`

resource_id

tag

octavia.db.healthcheck module

check_database_connection(*session*)
This is a simple database connection check function.

It will do a simple no-op query (low overhead) against the sqlalchemy session passed in.

Parameters **session** -- A Sql Alchemy database session.

Returns True if the connection check is successful, False if not.

octavia.db.models module

class Algorithm(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *octavia.db.base_models.LookupTableMixin*

description

name

class Amphora(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *octavia.db.base_models.IdMixin*, `oslo_db.sqlalchemy.models.TimestampMixin`

cached_zone

cert_busy

cert_expiration

compute_flavor

compute_id

created_at

ha_ip

ha_port_id

id

image_id

lb_network_ip

load_balancer

load_balancer_id

role

status

updated_at

vrrp_id

vrrp_interface

vrrp_ip

vrrp_port_id

vrrp_priority

class AmphoraBuildRequest(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

amphora_id

created_time

priority

status

class **AmphoraBuildSlots**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

id

slots_used

class **AmphoraHealth**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

amphora_id

busy

last_update

class **AmphoraRoles**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.LookupTableMixin*

description

name

class **AvailabilityZone**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.NameMixin*

availability_zone_profile_id

description

enabled

name

class **AvailabilityZoneProfile**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.IdMixin*, *octavia.db.base_models.NameMixin*

availability_zone_data

id

name

provider_name

class **ClientAuthenticationMode**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base

name

class **Flavor**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.IdMixin*, *octavia.db.base_models.NameMixin*

description

enabled

flavor_profile_id

id

name

`class` **`FlavorProfile`**`(`*`**kwargs`*`)`

Bases: `sqlalchemy.orm.decl_api.Base`, *`octavia.db.base_models.IdMixin`*, *`octavia.db.base_models.NameMixin`*

`flavor_data`

`id`

`name`

`provider_name`

`class` **`HealthMonitor`**`(`*`**kwargs`*`)`

Bases: `sqlalchemy.orm.decl_api.Base`, *`octavia.db.base_models.IdMixin`*, *`octavia.db.base_models.ProjectMixin`*, `oslo_db.sqlalchemy.models.TimestampMixin`, *`octavia.db.base_models.NameMixin`*, *`octavia.db.base_models.TagMixin`*

`created_at`

`delay`

`domain_name`

`enabled`

`expected_codes`

`fall_threshold`

`http_method`

`http_version`

`id`

`name`

`operating_status`

`pool`

`pool_id`

`project_id`

`provisioning_status`

`rise_threshold`

`timeout`

`type`

`updated_at`

`url_path`

`class` **`HealthMonitorType`**`(`*`**kwargs`*`)`

Bases: `sqlalchemy.orm.decl_api.Base`, *`octavia.db.base_models.LookupTableMixin`*

`description`

`name`

class L7Policy(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *`octavia.db.base_models.IdMixin`*, *`octavia.db.base_models.ProjectMixin`*, `oslo_db.sqlalchemy.models.TimestampMixin`, *`octavia.db.base_models.NameMixin`*, *`octavia.db.base_models.TagMixin`*

action

created_at

description

enabled

id

l7rules

listener

listener_id

name

operating_status

position

project_id

provisioning_status

redirect_http_code

redirect_pool

redirect_pool_id

redirect_prefix

redirect_url

updated_at

class L7PolicyAction(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *`octavia.db.base_models.LookupTableMixin`*

description

name

class L7Rule(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *`octavia.db.base_models.IdMixin`*, *`octavia.db.base_models.ProjectMixin`*, `oslo_db.sqlalchemy.models.TimestampMixin`, *`octavia.db.base_models.TagMixin`*

compare_type

created_at

enabled

id

invert

key

l7policy

l7policy_id

operating_status

project_id

provisioning_status

type

updated_at

value

class **L7RuleCompareType**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.LookupTableMixin*

description

name

class **L7RuleType**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.LookupTableMixin*

description

name

class **LBTopology**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.LookupTableMixin*

description

name

class **Listener**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.IdMixin*, *octavia.db.base_models.ProjectMixin*, oslo_db.sqlalchemy.models.TimestampMixin, *octavia.db.base_models.NameMixin*, *octavia.db.base_models.TagMixin*

allowed_cidrs

alpn_protocols

client_authentication

client_ca_tls_certificate_id

client_crl_container_id

connection_limit

created_at

default_pool

default_pool_id

description

enabled

id

insert_headers

l7policies

load_balancer

load_balancer_id

name

operating_status

peer_port

property **pools**

project_id

protocol

protocol_port

provisioning_status

sni_containers

timeout_client_data

timeout_member_connect

timeout_member_data

timeout_tcp_inspect

tls_certificate_id

tls_ciphers

tls_versions

updated_at

class **ListenerCidr**(***kwargs*)
Bases: `sqlalchemy.orm.decl_api.Base`

cidr

listener

listener_id

class **ListenerStatistics**(***kwargs*)
Bases: `sqlalchemy.orm.decl_api.Base`

active_connections

amphora_id

bytes_in

bytes_out

listener_id

request_errors

total_connections

validate_non_negative_int(*key*, *value*)

class **LoadBalancer**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.IdMixin*, *octavia.db.base_models.ProjectMixin*, oslo_db.sqlalchemy.models.TimestampMixin, *octavia.db.base_models.NameMixin*, *octavia.db.base_models.TagMixin*

amphorae

availability_zone

created_at

description

enabled

flavor_id

id

listeners

name

operating_status

pools

project_id

provider

provisioning_status

server_group_id

topology

updated_at

vip

vrrp_group

class **Member**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.IdMixin*, *octavia.db.base_models.ProjectMixin*, oslo_db.sqlalchemy.models.TimestampMixin, *octavia.db.base_models.NameMixin*, *octavia.db.base_models.TagMixin*

backup

created_at

enabled

id

ip_address

monitor_address

monitor_port

name

operating_status

pool

pool_id

project_id

protocol_port

provisioning_status

subnet_id

updated_at

weight

class **OperatingStatus**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.LookupTableMixin*

description

name

class **Pool**(***kwargs*)

Bases: sqlalchemy.orm.decl_api.Base, *octavia.db.base_models.IdMixin*, *octavia.db.base_models.ProjectMixin*, oslo_db.sqlalchemy.models.TimestampMixin, *octavia.db.base_models.NameMixin*, *octavia.db.base_models.TagMixin*

alpn_protocols

ca_tls_certificate_id

created_at

crl_container_id

description

enabled

health_monitor

id

l7policies

lb_algorithm

property **listeners**

load_balancer

load_balancer_id

members

name

operating_status

project_id

protocol

provisioning_status

session_persistence

tls_certificate_id

tls_ciphers

tls_enabled

tls_versions

updated_at

class **Protocol**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *octavia.db.base_models.LookupTableMixin*

description

name

class **ProvisioningStatus**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *octavia.db.base_models.LookupTableMixin*

description

name

class **Quotas**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

health_monitor

in_use_health_monitor

in_use_l7policy

in_use_l7rule

in_use_listener

in_use_load_balancer

in_use_member

in_use_pool

l7policy

l7rule

listener

load_balancer

member

pool

project_id

class **SNI**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

listener

listener_id

position

tls_container_id

class **SessionPersistence**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

cookie_name

persistence_granularity

persistence_timeout

pool

pool_id

type

class **SessionPersistenceType**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *octavia.db.base_models.LookupTableMixin*

description

name

class **VRRPAuthMethod**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`, *octavia.db.base_models.LookupTableMixin*

description

name

class **VRRPGroup**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

advert_int

load_balancer

load_balancer_id

vrrp_auth_pass

vrrp_auth_type

vrrp_group_name

class **Vip**(***kwargs*)

Bases: `sqlalchemy.orm.decl_api.Base`

ip_address

load_balancer

load_balancer_id

network_id

octavia_owned

port_id

qos_policy_id

subnet_id

octavia.db.prepare module

create_health_monitor(*hm_dict*, *pool_id=None*)

create_l7policy(*l7policy_dict*, *lb_id*, *listener_id*)

create_l7rule(*l7rule_dict*, *l7policy_id*)

create_listener(*listener_dict*, *lb_id*)

create_load_balancer(*lb_dict*)

create_load_balancer_tree(*lb_dict*)

create_member(*member_dict*, *pool_id*, *has_health_monitor=False*)

create_pool(*pool_dict*, *lb_id=None*)

octavia.db.repositories module

Defines interface for DB access that Resource or Octavia Controllers may reference

class **AmphoraBuildReqRepository**

Bases: `octavia.db.repositories.BaseRepository`

add_to_build_queue(*session*, *amphora_id=None*, *priority=None*)

Adds the build request to the table.

delete_all(*session*)

Deletes all the build requests.

get_highest_priority_build_req(*session*)

Fetches build request with highest priority and least created_time.

priority 20 = failover (highest) priority 40 = create_loadbalancer (lowest) :param session: A Sql Alchemy database session. :returns amphora_id corresponding to highest priority and least created time in 'WAITING' status.

model_class

alias of `octavia.db.models.AmphoraBuildRequest`

update_req_status(*session*, *amphora_id=None*)

Updates the request status.

class **AmphoraBuildSlotsRepository**

Bases: `octavia.db.repositories.BaseRepository`

get_used_build_slots_count(*session*)

Gets the number of build slots in use.

Returns Number of current build slots.

model_class

alias of `octavia.db.models.AmphoraBuildSlots`

update_count(*session*, *action='increment'*)

Increments/Decrements/Resets the number of build_slots used.

class **AmphoraHealthRepository**

Bases: `octavia.db.repositories.BaseRepository`

check_amphora_health_expired(*session*, *amphora_id*, *exp_age=None*)
check if a specific amphora is expired in the amphora_health table

Parameters

- **session** -- A Sql Alchemy database session.
- **amphora_id** -- id of an amphora object
- **exp_age** -- A standard datetime delta which is used to see for how long can an amphora live without updates before it is considered expired (default: CONF.house_keeping.amphora_expiry_age)

Returns boolean

get_stale_amphora(*session*)
Retrieves a stale amphora from the health manager database.

Parameters **session** -- A Sql Alchemy database session.

Returns [octavia.common.data_model]

model_class
alias of `octavia.db.models.AmphoraHealth`

replace(*session*, *amphora_id*, ***model_kwargs*)
replace or insert amphora into database.

update(*session*, *amphora_id*, ***model_kwargs*)
Updates a healthmanager entity in the database by amphora_id.

class **AmphoraRepository**
Bases: `octavia.db.repositories.BaseRepository`

allocate_and_associate(*session*, *load_balancer_id*, *availability_zone=None*)
Allocate an amphora for a load balancer.

For v0.5 this is simple, find a free amp and associate the lb. In the future this needs to be enhanced.

Parameters

- **session** -- A Sql Alchemy database session.
- **load_balancer_id** -- The load balancer id to associate

Returns The amphora ID for the load balancer or None

associate(*session*, *load_balancer_id*, *amphora_id*)
Associates an amphora with a load balancer.

Parameters

- **session** -- A Sql Alchemy database session.
- **load_balancer_id** -- The load balancer id to associate
- **amphora_id** -- The amphora id to associate

get_all_API_list(*session*, *pagination_helper=None*, ***filters*)
Get a list of amphorae for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API amphora list method.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`pagination_helper`** -- Helper to apply pagination and sorting.
- **`filters`** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

`get_cert_expiring_amphora`(*session*)

Retrieves an amphora whose cert is close to expiring..

Parameters **`session`** -- A Sql Alchemy database session.

Returns one amphora with expiring certificate

`static` **`get_lb_for_amphora`**(*session*, *amphora_id*)

Get all of the load balancers on an amphora.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`amphora_id`** -- The amphora id to list the load balancers from

Returns [octavia.common.data_model]

`get_lb_for_health_update`(*session*, *amphora_id*)

This method is for the health manager status update process.

This is a time sensitive query that occurs often. It is an explicit query as the ORM produces a poorly optimized query.

Use extreme caution making any changes to this query as it can impact the scalability of the health manager. All changes should be analyzed using SQL "EXPLAIN" to make sure only indexes are being used. Changes should also be evaluated using the stressHM tool.

Note: The returned object is flat and not a graph representation of the load balancer as it is not needed. This is on purpose to optimize the processing time. This is not in the normal data model objects.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`amphora_id`** -- The amphora ID to lookup the load balancer for.

Returns A dictionary containing the required load balancer details.

`model_class`

alias of *`octavia.db.models.Amphora`*

`test_and_set_status_for_delete`(*lock_session*, *id*)

Tests and sets an amphora status.

Puts a lock on the amphora table to check the status of the amphora. The status must be either AMPHORA_READY or ERROR to successfuly update the amphora status.

Parameters

- **`lock_session`** -- A Sql Alchemy database session.

- **`id`** -- id of Load Balancer

Raises

- *`ImmutableObject`* -- The amphora is not in a state that can be deleted.
- **`NoResultFound`** -- The amphora was not found or already deleted.

Returns None

`class AvailabilityZoneProfileRepository`

Bases: `octavia.db.repositories._GetALLExceptDELETEDIdMixin`, *`octavia.db.repositories.BaseRepository`*

`model_class`

alias of *`octavia.db.models.AvailabilityZoneProfile`*

`class AvailabilityZoneRepository`

Bases: `octavia.db.repositories._GetALLExceptDELETEDIdMixin`, *`octavia.db.repositories.BaseRepository`*

`delete`(*serial_session*, ***filters*)

Special delete method for availability_zone.

Sets DELETED LBs availability_zone to NIL_UUID, then removes the availability_zone.

Parameters

- **`serial_session`** -- A Sql Alchemy database transaction session.
- **`filters`** -- Filters to decide which entity should be deleted.

Returns None

Raises odb_exceptions.DBReferenceError

Raises sqlalchemy.orm.exc.NoResultFound

`get_availability_zone_metadata_dict`(*session*, *availability_zone_name*)

`get_availability_zone_provider`(*session*, *availability_zone_name*)

`model_class`

alias of *`octavia.db.models.AvailabilityZone`*

`update`(*session*, *name*, ***model_kwargs*)

Updates an entity in the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`model_kwargs`** -- Entity attributes that should be updates.

Returns octavia.common.data_model

`class BaseRepository`

Bases: `object`

`count`(*session*, ***filters*)

Retrieves a count of entities from the database.

Parameters

- **`session`** -- A Sql Alchemy database session.

- **`filters`** -- Filters to decide which entities should be retrieved.

Returns int

`create`(*session*, ***model_kwargs*)
Base create method for a database entity.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`model_kwargs`** -- Attributes of the model to insert.

Returns octavia.common.data_model

`delete`(*session*, ***filters*)
Deletes an entity from the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`filters`** -- Filters to decide which entity should be deleted.

Returns None

Raises sqlalchemy.orm.exc.NoResultFound

`delete_batch`(*session*, *ids=None*)
Batch deletes by entity ids.

`exists`(*session*, *id*)
Determines whether an entity exists in the database by its id.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`id`** -- id of entity to check for existence.

Returns octavia.common.data_model

`get`(*session*, ***filters*)
Retrieves an entity from the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`filters`** -- Filters to decide which entity should be retrieved.

Returns octavia.common.data_model

`get_all`(*session*, *pagination_helper=None*, *query_options=None*, ***filters*)
Retrieves a list of entities from the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`pagination_helper`** -- Helper to apply pagination and sorting.
- **`query_options`** -- Optional query options to apply.
- **`filters`** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

get_all_deleted_expiring(*session*, *exp_age*)
Get all previously deleted resources that are now expiring.

Parameters

- **session** -- A Sql Alchemy database session.
- **exp_age** -- A standard datetime delta which is used to see for how long can a resource live without updates before it is considered expired

Returns A list of resource IDs

model_class = None

update(*session*, *id*, ***model_kwargs*)
Updates an entity in the database.

Parameters

- **session** -- A Sql Alchemy database session.
- **model_kwargs** -- Entity attributes that should be updates.

Returns octavia.common.data_model

class FlavorProfileRepository
Bases: `octavia.db.repositories._GetALLExceptDELETEDIdMixin`, *octavia.db.repositories.BaseRepository*

model_class
alias of *octavia.db.models.FlavorProfile*

class FlavorRepository
Bases: `octavia.db.repositories._GetALLExceptDELETEDIdMixin`, *octavia.db.repositories.BaseRepository*

delete(*serial_session*, ***filters*)
Sets DELETED LBs flavor_id to NIL_UUID, then removes the flavor

Parameters

- **serial_session** -- A Sql Alchemy database transaction session.
- **filters** -- Filters to decide which entity should be deleted.

Returns None

Raises odb_exceptions.DBReferenceError

Raises sqlalchemy.orm.exc.NoResultFound

get_flavor_metadata_dict(*session*, *flavor_id*)

get_flavor_provider(*session*, *flavor_id*)

model_class
alias of *octavia.db.models.Flavor*

class HealthMonitorRepository
Bases: *octavia.db.repositories.BaseRepository*

`get_all_API_list`(*session*, *pagination_helper=None*, ***filters*)

Get a list of health monitors for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API health monitor list method.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`pagination_helper`** -- Helper to apply pagination and sorting.
- **`filters`** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

`model_class`

alias of *`octavia.db.models.HealthMonitor`*

`class L7PolicyRepository`

Bases: *`octavia.db.repositories.BaseRepository`*

`create`(*session*, ***model_kwargs*)

Base create method for a database entity.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`model_kwargs`** -- Attributes of the model to insert.

Returns octavia.common.data_model

`delete`(*session*, *id*, ***filters*)

Deletes an entity from the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`filters`** -- Filters to decide which entity should be deleted.

Returns None

Raises sqlalchemy.orm.exc.NoResultFound

`get_all`(*session*, *pagination_helper=None*, ***filters*)

Retrieves a list of entities from the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`pagination_helper`** -- Helper to apply pagination and sorting.
- **`query_options`** -- Optional query options to apply.
- **`filters`** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

`get_all_API_list`(*session*, *pagination_helper=None*, ***filters*)

`model_class`

alias of *`octavia.db.models.L7Policy`*

`update`(*session*, *id*, ***model_kwargs*)

Updates an entity in the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`model_kwargs`** -- Entity attributes that should be updates.

Returns octavia.common.data_model

`class L7RuleRepository`

Bases: *`octavia.db.repositories.BaseRepository`*

`create`(*session*, ***model_kwargs*)

Base create method for a database entity.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`model_kwargs`** -- Attributes of the model to insert.

Returns octavia.common.data_model

`get_all_API_list`(*session*, *pagination_helper=None*, ***filters*)

Get a list of L7 Rules for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API L7 Rule list method.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`pagination_helper`** -- Helper to apply pagination and sorting.
- **`filters`** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

`model_class`

alias of *`octavia.db.models.L7Rule`*

`update`(*session*, *id*, ***model_kwargs*)

Updates an entity in the database.

Parameters

- **`session`** -- A Sql Alchemy database session.
- **`model_kwargs`** -- Entity attributes that should be updates.

Returns octavia.common.data_model

`class ListenerCidrRepository`

Bases: *`octavia.db.repositories.BaseRepository`*

`create`(*session*, *listener_id*, *allowed_cidrs*)

Base create method for a database entity.

Parameters

- **`session`** -- A Sql Alchemy database session.

- **model_kwargs** -- Attributes of the model to insert.

Returns octavia.common.data_model

model_class
alias of *octavia.db.models.ListenerCidr*

update(*session*, *listener_id*, *allowed_cidrs*)
Updates allowed CIDRs in the database by listener_id.

class ListenerRepository
Bases: *octavia.db.repositories.BaseRepository*

create(*session*, ***model_kwargs*)
Creates a new Listener with some validation.

get_all_API_list(*session*, *pagination_helper=None*, ***filters*)
Get a list of listeners for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API listener list method.

Parameters

- **session** -- A Sql Alchemy database session.
- **pagination_helper** -- Helper to apply pagination and sorting.
- **filters** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

has_default_pool(*session*, *id*)
Checks if a listener has a default pool.

model_class
alias of *octavia.db.models.Listener*

prov_status_active_if_not_error(*session*, *listener_id*)
Update provisioning_status to ACTIVE if not already in ERROR.

update(*session*, *id*, ***model_kwargs*)
Updates an entity in the database.

Parameters

- **session** -- A Sql Alchemy database session.
- **model_kwargs** -- Entity attributes that should be updates.

Returns octavia.common.data_model

class ListenerStatisticsRepository
Bases: *octavia.db.repositories.BaseRepository*

increment(*session*, *delta_stats*)
Updates a listener's statistics, incrementing by the passed deltas.

Parameters

- **session** -- A Sql Alchemy database session
- **delta_stats** (`octavia.common.data_models.ListenerStatistics`) -- Listener statistics deltas to add

model_class
alias of *octavia.db.models.ListenerStatistics*

replace(*session*, *stats_obj*)
Create or override a listener's statistics (insert/update)

Parameters

- **session** -- A Sql Alchemy database session
- **stats_obj** (octavia.common.data_models.ListenerStatistics) -- Listener statistics object to store

update(*session*, *listener_id*, ***model_kwargs*)
Updates a listener's statistics, overriding with the passed values.

Parameters

- **session** -- A Sql Alchemy database session
- **listener_id** (*str*) -- The UUID of the listener to update
- **model_kwargs** -- Entity attributes that should be updated

class LoadBalancerRepository
Bases: *octavia.db.repositories.BaseRepository*

get_all_API_list(*session*, *pagination_helper=None*, ***filters*)
Get a list of load balancers for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API load balancer list method.

Parameters

- **session** -- A Sql Alchemy database session.
- **pagination_helper** -- Helper to apply pagination and sorting.
- **filters** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

model_class
alias of *octavia.db.models.LoadBalancer*

set_status_for_failover(*session*, *id*, *status*, *raise_exception=False*)
Tests and sets a load balancer provisioning status.

Puts a lock on the load balancer table to check the status of a load balancer. If the status is ACTIVE or ERROR then the status of the load balancer is updated and the method returns True. If the status is not ACTIVE, then nothing is done and False is returned.

Parameters

- **session** -- A Sql Alchemy database session.
- **id** -- id of Load Balancer
- **status** -- Status to set Load Balancer if check passes.
- **raise_exception** -- If True, raise ImmutableObject on failure

Returns bool

test_and_set_provisioning_status(*session*, *id*, *status*, *raise_exception=False*)
Tests and sets a load balancer and provisioning status.

Puts a lock on the load balancer table to check the status of a load balancer. If the status is ACTIVE then the status of the load balancer is updated and the method returns True. If the status is not ACTIVE, then nothing is done and False is returned.

Parameters

- **session** -- A Sql Alchemy database session.
- **id** -- id of Load Balancer
- **status** -- Status to set Load Balancer if check passes.
- **raise_exception** -- If True, raise ImmutableObject on failure

Returns bool

class MemberRepository
Bases: `octavia.db.repositories.BaseRepository`

delete_members(*session*, *member_ids*)
Batch deletes members from a pool.

get_all_API_list(*session*, *pagination_helper=None*, ***filters*)
Get a list of members for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API member list method.

Parameters

- **session** -- A Sql Alchemy database session.
- **pagination_helper** -- Helper to apply pagination and sorting.
- **filters** -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

model_class
alias of `octavia.db.models.Member`

update_pool_members(*session*, *pool_id*, ***model_kwargs*)
Updates all of the members of a pool.

Parameters

- **session** -- A Sql Alchemy database session.
- **pool_id** -- ID of the pool to update members on.
- **model_kwargs** -- Entity attributes that should be updates.

Returns octavia.common.data_model

class PoolRepository
Bases: `octavia.db.repositories.BaseRepository`

get_all_API_list(*session*, *pagination_helper=None*, ***filters*)
Get a list of pools for the API list call.

This get_all returns a data set that is only one level deep in the data graph. This is an optimized query for the API pool list method.

Parameters

- `session` -- A Sql Alchemy database session.
- `pagination_helper` -- Helper to apply pagination and sorting.
- `filters` -- Filters to decide which entities should be retrieved.

Returns [octavia.common.data_model]

`get_children_count`(*session*, *pool_id*)

`model_class`
alias of *octavia.db.models.Pool*

class QuotasRepository
Bases: *octavia.db.repositories.BaseRepository*

`delete`(*session*, *project_id*)
Deletes an entity from the database.

Parameters

- `session` -- A Sql Alchemy database session.
- `filters` -- Filters to decide which entity should be deleted.

Returns None

Raises sqlalchemy.orm.exc.NoResultFound

`model_class`
alias of *octavia.db.models.Quotas*

`update`(*session*, *project_id*, ***model_kwargs*)
Updates an entity in the database.

Parameters

- `session` -- A Sql Alchemy database session.
- `model_kwargs` -- Entity attributes that should be updates.

Returns octavia.common.data_model

class Repositories
Bases: `object`

`check_quota_met`(*session*, *lock_session*, *_class*, *project_id*, *count=1*)
Checks and updates object quotas.

This method makes sure the project has available quota for the resource and updates the quota to reflect the new ussage.

Parameters

- `session` -- Context database session
- `lock_session` -- Locking database session (autocommit=False)
- `_class` -- Data model object requesting quota

- `project_id` -- Project ID requesting quota
- `count` -- Number of objects we're going to create (default=1)

Returns True if quota is met, False if quota was available

`create_load_balancer_and_vip`(*session*, *lb_dict*, *vip_dict*)

Inserts load balancer and vip entities into the database.

Inserts load balancer and vip entities into the database in one transaction and returns the data model of the load balancer.

Parameters

- `session` -- A Sql Alchemy database session.
- `lb_dict` -- Dictionary representation of a load balancer
- `vip_dict` -- Dictionary representation of a vip

Returns octavia.common.data_models.LoadBalancer

`create_load_balancer_tree`(*session*, *lock_session*, *lb_dict*)

`create_pool_on_load_balancer`(*session*, *pool_dict*, *listener_id=None*)

Inserts a pool and session persistence entity into the database.

Parameters

- `session` -- A Sql Alchemy database session.
- `pool_dict` -- Dictionary representation of a pool
- `listener_id` -- Optional listener id that will reference this pool as its default_pool_id

Returns octavia.common.data_models.Pool

`decrement_quota`(*lock_session*, *_class*, *project_id*, *quantity=1*)

Decrements the object quota for a project

Parameters

- `lock_session` -- Locking database session (autocommit=False)
- `_class` -- Data model object to decrement quota
- `project_id` -- Project ID to decrement quota on
- `quantity` -- Quantity of quota to decrement

Returns None

`get_amphora_stats`(*session*, *amp_id*)

Gets the statistics for all listeners on an amphora.

Parameters

- `session` -- A Sql Alchemy database session.
- `amp_id` -- The amphora ID to query.

Returns An amphora stats dictionary

test_and_set_lb_and_listeners_prov_status(*session*, *lb_id*, *lb_prov_status*, *listener_prov_status*, *listener_ids=None*, *pool_id=None*, *l7policy_id=None*)

Tests and sets a load balancer and listener provisioning status.

Puts a lock on the load balancer table to check the status of a load balancer. If the status is ACTIVE then the status of the load balancer and listener is updated and the method returns True. If the status is not ACTIVE, then nothing is done and False is returned.

Parameters

- **session** -- A Sql Alchemy database session.
- **lb_id** -- ID of the Load Balancer to check and lock
- **lb_prov_status** -- Status to set Load Balancer and Listener if check passes.
- **listener_prov_status** -- Status to set Listeners if check passes
- **listener_ids** -- List of IDs of listeners to check and lock (only use this when relevant to the operation)
- **pool_id** -- ID of the Pool to check and lock (only use this when relevant to the operation)
- **l7policy_id** -- ID of the L7Policy to check and lock (only use this when relevant to the operation)

Returns bool

update_pool_and_sp(*session*, *pool_id*, *pool_dict*)

Updates a pool and session persistence entity in the database.

Parameters

- **session** -- A Sql Alchemy database session.
- **pool_dict** -- Dictionary representation of a pool

Returns octavia.common.data_models.Pool

class SNIRepository

Bases: *octavia.db.repositories.BaseRepository*

model_class

alias of *octavia.db.models.SNI*

update(*session*, *listener_id=None*, *tls_container_id=None*, ***model_kwargs*)

Updates an SNI entity in the database.

class SessionPersistenceRepository

Bases: *octavia.db.repositories.BaseRepository*

exists(*session*, *pool_id*)

Checks if session persistence exists on a pool.

model_class

alias of *octavia.db.models.SessionPersistence*

update(*session*, *pool_id*, ***model_kwargs*)

Updates a session persistence entity in the database by pool_id.

class VRRPGroupRepository

Bases: `octavia.db.repositories.BaseRepository`

model_class

alias of `octavia.db.models.VRRPGroup`

update(*session*, *load_balancer_id*, ***model_kwargs*)

Updates a VRRPGroup entry for by load_balancer_id.

class VipRepository

Bases: `octavia.db.repositories.BaseRepository`

model_class

alias of `octavia.db.models.Vip`

update(*session*, *load_balancer_id*, ***model_kwargs*)

Updates a vip entity in the database by load_balancer_id.

Module contents

octavia.distributor package

Subpackages

octavia.distributor.drivers package

Subpackages

octavia.distributor.drivers.noop_driver package

Submodules

octavia.distributor.drivers.noop_driver.driver module

class NoopDistributorDriver

Bases: `octavia.distributor.drivers.driver_base.DistributorDriver`

get_add_vip_subflow()

Get a subflow that adds a VIP to a distributor

Requires **distributor_id** (string) - The ID of the distributor to create the VIP on.

Requires **vip** (object) - The VIP object to create on the distributor.

Requires **vip_alg** (string) - The optional algorithm to use for this VIP.

Requires **vip_persistence** (string) - The persistence type for this VIP.

Returns A TaskFlow Flow that will add a VIP to the distributor

This method will return a TaskFlow Flow that adds a VIP to the distributor by perfoming the necessary steps to plug the VIP and configure the distributor to start receiving requests on this VIP.

`get_create_distributor_subflow()`

Get a subflow to create a distributor

Requires load_balancer (object) - Load balancer object associated with this distributor

Provides distributor_id (string) - The created distributor ID

Returns A TaskFlow Flow that will create the distributor

This method will setup the TaskFlow Flow required to setup the database fields and create a distributor should the driver need to instantiate one. The flow must store the generated distibutor ID in the flow.

`get_delete_distributor_subflow()`

Get a subflow that deletes a distributor

Requires distributor_id (string) - The ID of the distributor to delete

Returns A TaskFlow Flow that will delete the distributor

This method will return a TaskFlow Flow that deletes the distributor (if applicable for the driver) and cleans up any associated database records.

`get_drain_amphorae_subflow()`

Get a subflow that drains connections from amphorae

Requires distributor_id (string) - The ID of the distributor to drain amphorae from

Requires amphorae (tuple) - Tuple of amphora objects to drain from distributor.

Returns A TaskFlow Flow that will drain the listed amphorae on the distributor

This method will return a TaskFlow Flow that configures the distributor to stop sending new connections to the amphorae in the list. Existing connections will continue to pass traffic to the amphorae in this list.

`get_register_amphorae_subflow()`

Get a subflow that Registers amphorae with the distributor

Requires distributor_id (string) - The ID of the distributor to register the amphora on

Requires amphorae (tuple) - Tuple of amphora objects to register with the distributor.

Returns A TaskFlow Flow that will register amphorae with the distributor

This method will return a TaskFlow Flow that registers amphorae with the distributor so it can begin to receive requests from the distributor. Amphora should be ready to receive requests prior to this call being made.

`get_remove_vip_subflow()`

Get a subflow that removes a VIP from a distributor

Requires distributor_id (string) - The ID of the distributor to remove the VIP from.

Requires vip (object) - The VIP object to remove from the distributor.

Returns A TaskFlow Flow that will remove a VIP from the distributor

This method will return a TaskFlow Flow that removes the VIP from the distributor by reconfiguring the distributor and unplugging the associated port.

get_unregister_amphorae_subflow()

Get a subflow that unregisters amphorae from a distributor

Requires distributor_id (string) - The ID of the distributor to unregister amphorae from

Requires amphorae (tuple) - Tuple of amphora objects to unregister from distributor.

Returns A TaskFlow Flow that will unregister amphorae from the distributor

This method will return a TaskFlow Flow that unregisters amphorae from the distributor. Amphorae in this list will immediately stop receiving traffic.

class **NoopManager**

Bases: `object`

get_add_vip_subflow()

get_create_distributor_subflow()

get_delete_distributor_subflow()

get_drain_amphorae_subflow()

get_register_amphorae_subflow()

get_remove_vip_subflow()

get_unregister_amphorae_subflow()

class **NoopProvidesRequiresTask**(*name*, *provides_dicts=None*, *requires=None*)

Bases: `taskflow.task.Task`

execute(**args*, ***kwargs*)

Activate a given atom which will perform some operation and return.

This method can be used to perform an action on a given set of input requirements (passed in via `*args` and `**kwargs`) to accomplish some type of operation. This operation may provide some named outputs/results as a result of it executing for later reverting (or for other atoms to depend on).

NOTE(harlowja): the result (if any) that is returned should be persistable so that it can be passed back into this atom if reverting is triggered (especially in the case where reverting happens in a different python process or on a remote machine) and so that the result can be transmitted to other atoms (which may be local or remote).

Parameters

- **args** -- positional arguments that atom requires to execute.
- **kwargs** -- any keyword arguments that atom requires to execute.

Module contents

Submodules

octavia.distributor.drivers.driver_base module

class DistributorDriver

Bases: `object`

abstract get_add_vip_subflow()

Get a subflow that adds a VIP to a distributor

Requires **distributor_id** (string) - The ID of the distributor to create the VIP on.

Requires **vip** (object) - The VIP object to create on the distributor.

Requires **vip_alg** (string) - The optional algorithm to use for this VIP.

Requires **vip_persistence** (string) - The persistence type for this VIP.

Returns A TaskFlow Flow that will add a VIP to the distributor

This method will return a TaskFlow Flow that adds a VIP to the distributor by perfoming the necessary steps to plug the VIP and configure the distributor to start receiving requests on this VIP.

abstract get_create_distributor_subflow()

Get a subflow to create a distributor

Requires **load_balancer** (object) - Load balancer object associated with this distributor

Provides **distributor_id** (string) - The created distributor ID

Returns A TaskFlow Flow that will create the distributor

This method will setup the TaskFlow Flow required to setup the database fields and create a distributor should the driver need to instantiate one. The flow must store the generated distibutor ID in the flow.

abstract get_delete_distributor_subflow()

Get a subflow that deletes a distributor

Requires **distributor_id** (string) - The ID of the distributor to delete

Returns A TaskFlow Flow that will delete the distributor

This method will return a TaskFlow Flow that deletes the distributor (if applicable for the driver) and cleans up any associated database records.

abstract get_drain_amphorae_subflow()

Get a subflow that drains connections from amphorae

Requires **distributor_id** (string) - The ID of the distributor to drain amphorae from

Requires **amphorae** (tuple) - Tuple of amphora objects to drain from distributor.

Returns A TaskFlow Flow that will drain the listed amphorae on the distributor

This method will return a TaskFlow Flow that configures the distributor to stop sending new connections to the amphorae in the list. Existing connections will continue to pass traffic to the amphorae in this list.

`abstract get_register_amphorae_subflow()`

Get a subflow that Registers amphorae with the distributor

Requires **distributor_id** (string) - The ID of the distributor to register the amphora on

Requires **amphorae** (tuple) - Tuple of amphora objects to register with the distributor.

Returns A TaskFlow Flow that will register amphorae with the distributor

This method will return a TaskFlow Flow that registers amphorae with the distributor so it can begin to receive requests from the distributor. Amphora should be ready to receive requests prior to this call being made.

`abstract get_remove_vip_subflow()`

Get a subflow that removes a VIP from a distributor

Requires **distributor_id** (string) - The ID of the distributor to remove the VIP from.

Requires **vip** (object) - The VIP object to remove from the distributor.

Returns A TaskFlow Flow that will remove a VIP from the distributor

This method will return a TaskFlow Flow that removes the VIP from the distributor by reconfiguring the distributor and unplugging the associated port.

`abstract get_unregister_amphorae_subflow()`

Get a subflow that unregisters amphorae from a distributor

Requires **distributor_id** (string) - The ID of the distributor to unregister amphorae from

Requires **amphorae** (tuple) - Tuple of amphora objects to unregister from distributor.

Returns A TaskFlow Flow that will unregister amphorae from the distributor

This method will return a TaskFlow Flow that unregisters amphorae from the distributor. Amphorae in this list will immediately stop receiving traffic.

Module contents

Module contents

octavia.hacking package

Submodules

octavia.hacking.checks module

Guidelines for writing new hacking checks

- Use only for Octavia specific tests. OpenStack general tests should be submitted to the common 'hacking' module.
- Pick numbers in the range O3xx. Find the current test with the highest allocated number and then pick the next value.
- Keep the test method code in the source file ordered based on the O3xx value.
- List the new rule in the top level HACKING.rst file
- Add test cases for each new rule to octavia/tests/unit/test_hacking.py

`assert_equal_in`(*logical_line*)

Check for assertEqual(A in B, True), assertEqual(True, A in B),

assertEqual(A in B, False) or assertEqual(False, A in B) sentences

O338

`assert_equal_or_not_none`(*logical_line*)

Check for assertEqual(A, None) or assertEqual(None, A) sentences,

assertNotEqual(A, None) or assertNotEqual(None, A) sentences

O318

`assert_equal_true_or_false`(*logical_line*)

Check for assertEqual(True, A) or assertEqual(False, A) sentences

O323

`assert_true_instance`(*logical_line*)

Check for assertTrue(isinstance(a, b)) sentences

O316

`check_line_continuation_no_backslash`(*logical_line*, *tokens*)

O346 - Don't use backslashes for line continuation.

Parameters

- **`logical_line`** -- The logical line to check. Not actually used.
- **`tokens`** -- List of tokens to check.

Returns None if the tokens don't contain any issues, otherwise a tuple is yielded that contains the offending index in the logical line and a message describe the check validation failure.

`check_no_eventlet_imports`(*logical_line*)

O345 - Usage of Python eventlet module not allowed.

Parameters **`logical_line`** -- The logical line to check.

Returns None if the logical line passes the check, otherwise a tuple is yielded that contains the offending index in logical line and a message describe the check validation failure.

`check_no_logging_imports`(*logical_line*)

O348 - Usage of Python logging module not allowed.

Parameters **`logical_line`** -- The logical line to check.

Returns None if the logical line passes the check, otherwise a tuple is yielded that contains the offending index in logical line and a message describe the check validation failure.

`check_raised_localized_exceptions`(*logical_line*, *filename*)

O342 - Untranslated exception message.

Parameters

- `logical_line` -- The logical line to check.
- `filename` -- The file name where the logical line exists.

Returns None if the logical line passes the check, otherwise a tuple is yielded that contains the offending index in logical line and a message describe the check validation failure.

`no_log_warn`(*logical_line*)

Disallow 'LOG.warn('

O339

`no_mutable_default_args`(*logical_line*)

`no_translate_logs`(*logical_line*, *filename*)

O341 - Don't translate logs.

Check for 'LOG.*(_(' and 'LOG.*(_Lx('

Translators don't provide translations for log messages, and operators asked not to translate them.

- This check assumes that 'LOG' is a logger.

Parameters

- `logical_line` -- The logical line to check.
- `filename` -- The file name where the logical line exists.

Returns None if the logical line passes the check, otherwise a tuple is yielded that contains the offending index in logical line and a message describe the check validation failure.

`revert_must_have_kwargs`(*logical_line*)

O347 - Taskflow revert methods must have **kwargs.

Parameters `logical_line` -- The logical line to check.

Returns None if the logical line passes the check, otherwise a tuple is yielded that contains the offending index in logical line and a message describe the check validation failure.

Module contents

octavia.image package

Subpackages

octavia.image.drivers package

Subpackages

octavia.image.drivers.noop_driver package

Submodules

octavia.image.drivers.noop_driver.driver module

class NoopImageDriver

Bases: *octavia.image.image_base.ImageBase*

get_image_id_by_tag(*image_tag*, *image_owner=None*)

Get image ID by image tag and owner.

Parameters

- **image_tag** -- image tag
- **image_owner** -- optional image owner

Raises ImageGetException if no images found with given tag

Returns image id

class NoopManager

Bases: `object`

get_image_id_by_tag(*image_tag*, *image_owner=None*)

Module contents

Submodules

octavia.image.drivers.glance_driver module

class ImageManager

Bases: *octavia.image.image_base.ImageBase*

Image implementation of virtual machines via Glance.

get_image_id_by_tag(*image_tag*, *image_owner=None*)

Get image ID by image tag and owner

Parameters

- **`image_tag`** -- image tag
- **`image_owner`** -- optional image owner

Raises ImageGetException if no images found with given tag

Returns image id

Module contents

Submodules

octavia.image.image_base module

`class ImageBase`

Bases: `object`

`abstract get_image_id_by_tag``(`*image_tag*, *image_owner=None*`)`

Get image ID by image tag and owner.

Parameters

- **`image_tag`** -- image tag
- **`image_owner`** -- optional image owner

Raises ImageGetException if no images found with given tag

Returns image id

Module contents

octavia.network package

Subpackages

octavia.network.drivers package

Subpackages

octavia.network.drivers.neutron package

Submodules

octavia.network.drivers.neutron.allowed_address_pairs module

`class AllowedAddressPairsDriver`

Bases: *`octavia.network.drivers.neutron.base.BaseNeutronDriver`*

`allocate_vip``(`*load_balancer*`)`

Allocates a virtual ip.

Reserves it for later use as the frontend connection of a load balancer.

Parameters **`load_balancer`** -- octavia.common.data_models.LoadBalancer instance

Returns octavia.common.data_models.VIP

Raises AllocateVIPException, PortNotFound, SubnetNotFound

`create_port`(*network_id*, *name=None*, *fixed_ips=()*, *secondary_ips=()*, *security_group_ids=()*, *admin_state_up=True*, *qos_policy_id=None*)

Creates a network port.

fixed_ips = [{'subnet_id': <id>, ('ip_addrss': <IP>')},] ip_address is optional in the fixed_ips dictionary.

Parameters

- **`network_id`** -- The network the port should be created on.
- **`name`** -- The name to apply to the port.
- **`fixed_ips`** -- A list of fixed IP dicts.
- **`secondary_ips`** -- A list of secondary IPs to add to the port.
- **`security_group_ids`** -- A list of security group IDs for the port.
- **`qos_policy_id`** -- The QoS policy ID to apply to the port.

Returns port A port data model object.

`deallocate_vip`(*vip*)

Delete the vrrp_port (instance port) in case nova didn't

This can happen if a failover has occurred.

`delete_port`(*port_id*)

delete a neutron port.

Parameters **`port_id`** -- The port ID to delete.

Returns None

`failover_preparation`(*amphora*)

Prepare an amphora for failover.

Parameters **`amphora`** -- amphora object to failover

Returns None

Raises PortNotFound

`get_network_configs`(*loadbalancer*, *amphora=None*)

Retrieve network configurations

This method assumes that a dictionary of AmphoraNetworkConfigs keyed off of the related amphora id are returned. The configs contain data pertaining to each amphora that is later used for finalization of the entire load balancer configuration. The data provided to these configs is left up to the driver, this means the driver is responsible for providing data that is appropriate for the amphora network configurations.

Example return: {<amphora.id>: <AmphoraNetworkConfig>}

Parameters

- `load_balancer` -- The load_balancer configuration
- `amphora` -- Optional amphora to only query.

Returns dict of octavia.network.data_models.AmphoraNetworkConfig keyed off of the amphora id the config is associated with.

Raises NotFound, NetworkNotFound, SubnetNotFound, PortNotFound

`get_security_group`(*sg_name*)

Retrieves the security group by it's name.

Parameters `sg_name` -- The security group name.

Returns octavia.network.data_models.SecurityGroup, None if not enabled

Raises NetworkException, SecurityGroupNotFound

`plug_aap_port`(*load_balancer*, *vip*, *amphora*, *subnet*)

Plugs the AAP port to the amp

Parameters

- `load_balancer` -- Load Balancer to prepare the VIP for
- `vip` -- The VIP to plug
- `amphora` -- The amphora to plug the VIP into
- `subnet` -- The subnet to plug the aap into

`plug_network`(*compute_id*, *network_id*)

Connects an existing amphora to an existing network.

Parameters

- `compute_id` -- id of an amphora in the compute service
- `network_id` -- id of a network

Returns octavia.network.data_models.Interface instance

Raises PlugNetworkException, AmphoraNotFound, NetworkNotFound

`plug_port`(*amphora*, *port*)

Plug a neutron port in to a compute instance

Parameters

- `amphora` -- amphora object to plug the port into
- `port` -- port to plug into the compute instance

Returns None

Raises PlugNetworkException, AmphoraNotFound, NetworkNotFound

`plug_vip`(*load_balancer*, *vip*)

Plugs a virtual ip as the frontend connection of a load balancer.

Sets up the routing of traffic from the vip to the load balancer and its amphorae.

Parameters

- `load_balancer` -- octavia.common.data_models.LoadBalancer instance
- `vip` -- octavia.common.data_models.VIP instance

Returns dict consisting of amphora_id as key and bind_ip as value. bind_ip is the ip that the amphora should listen on to receive traffic to load balance.

Raises PlugVIPException, PortNotFound

`set_port_admin_state_up`(*port_id*, *state*)

Set the admin state of a port. True is up, False is down.

Parameters

- `port_id` -- The port ID to update.
- `state` -- True for up, False for down.

Returns None

`unplug_aap_port`(*vip*, *amphora*, *subnet*)

Unplugs the AAP port to the amp

Parameters

- `vip` -- The VIP to plug
- `amphora` -- The amphora to plug the VIP into
- `subnet` -- The subnet to plug the aap into

`unplug_network`(*compute_id*, *network_id*)

Disconnects an existing amphora from an existing network.

If ip_address is not specified, all the interfaces plugged on network_id should be unplugged.

Parameters

- `compute_id` -- id of an amphora in the compute service
- `network_id` -- id of a network

Returns None

Raises UnplugNetworkException, AmphoraNotFound, NetworkNotFound, NetworkException

`unplug_vip`(*load_balancer*, *vip*)

Unplugs a virtual ip as the frontend connection of a load balancer.

Removes the routing of traffic from the vip to the load balancer and its amphorae.

Parameters

- `load_balancer` -- octavia.common.data_models.LoadBalancer instance
- `vip` -- octavia.common.data_models.VIP instance

Returns octavia.common.data_models.VIP instance

Raises UnplugVIPException, PluggedVIPNotFound

`update_vip`(*load_balancer*, *for_delete=False*)

Hook for the driver to update the VIP information.

This method will be called upon the change of a load_balancer configuration. It is an optional method to be implemented by drivers. It allows the driver to update any VIP information based on the state of the passed in load_balancer.

Parameters

- **`load_balancer`** -- octavia.common.data_models.LoadBalancer instance
- **`for_delete`** -- Boolean indicating if this update is for a delete

Raises MissingVIPSecurityGroup

Returns None

`update_vip_sg`(*load_balancer*, *vip*)

Updates the security group for a VIP

Parameters

- **`load_balancer`** -- Load Balancer to rpepare the VIP for
- **`vip`** -- The VIP to plug

`wait_for_port_detach`(*amphora*)

Waits for the amphora ports device_id to be unset.

This method waits for the ports on an amphora device_id parameter to be ” or None which signifies that nova has finished detaching the port from the instance.

Parameters **`amphora`** -- Amphora to wait for ports to detach.

Returns None

Raises

- *`TimeoutException`* -- Port did not detach in interval.
- *`PortNotFound`* -- Port was not found by neutron.

octavia.network.drivers.neutron.base module

`class BaseNeutronDriver`

Bases: *`octavia.network.base.AbstractNetworkDriver`*

`apply_qos_on_port`(*qos_id*, *port_id*)

`get_network`(*network_id*, *context=None*)

Retrieves network from network id.

Parameters

- **`network_id`** -- id of an network to retrieve
- **`context`** -- A request context

Returns octavia.network.data_models.Network

Raises NetworkException, NetworkNotFound

`get_network_by_name`(*network_name*)

Retrieves network from network name.

Parameters **`network_name`** -- name of a network to retrieve

Returns octavia.network.data_models.Network

Raises NetworkException, NetworkNotFound

`get_network_ip_availability`(*network*)

Retrieves network IP availability.

Parameters `network` -- octavia.network.data_models.Network

Returns octavia.network.data_models.Network_IP_Availability

Raises NetworkException, NetworkNotFound

`get_plugged_networks`(*compute_id*)

Retrieves the current plugged networking configuration.

Parameters `compute_id` -- id of an amphora in the compute service

Returns [octavia.network.data_models.Instance]

`get_port`(*port_id*, *context=None*)

Retrieves port from port id.

Parameters

- `port_id` -- id of a port to retrieve
- `context` -- A request context

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`get_port_by_name`(*port_name*)

Retrieves port from port name.

Parameters `port_name` -- name of a port to retrieve

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`get_port_by_net_id_device_id`(*network_id*, *device_id*)

Retrieves port from network id and device id.

Parameters

- `network_id` -- id of a network to filter by
- `device_id` -- id of a network device to filter by

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`get_qos_policy`(*qos_policy_id*)

`get_subnet`(*subnet_id*, *context=None*)

Retrieves subnet from subnet id.

Parameters

- `subnet_id` -- id of a subnet to retrieve
- `context` -- A request context

Returns octavia.network.data_models.Subnet

Raises NetworkException, SubnetNotFound

`get_subnet_by_name`(*subnet_name*)

Retrieves subnet from subnet name.

Parameters `subnet_name` -- name of a subnet to retrieve

Returns octavia.network.data_models.Subnet

Raises NetworkException, SubnetNotFound

`plug_fixed_ip`(*port_id*, *subnet_id*, *ip_address=None*)

Plug a fixed ip to an existing port.

If ip_address is not specified, one will be auto-assigned.

Parameters

- `port_id` -- id of a port to add a fixed ip
- `subnet_id` -- id of a subnet
- `ip_address` -- specific ip_address to add

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`qos_enabled`()

Whether QoS is enabled

Returns Boolean

`unplug_fixed_ip`(*port_id*, *subnet_id*)

Unplug a fixed ip from an existing port.

Parameters

- `port_id` -- id of a port to remove the fixed ip from
- `subnet_id` -- id of a subnet

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

octavia.network.drivers.neutron.utils module

`convert_fixed_ip_dict_to_model`(*fixed_ip_dict*)

`convert_floatingip_dict_to_model`(*floating_ip_dict*)

`convert_network_dict_to_model`(*network_dict*)

`convert_network_ip_availability_dict_to_model`(*network_ip_availability_dict*)

`convert_port_dict_to_model`(*port_dict*)

`convert_qos_policy_dict_to_model`(*qos_policy_dict*)

`convert_security_group_dict_to_model`(*security_group_dict*)

convert_subnet_dict_to_model(*subnet_dict*)

Module contents

octavia.network.drivers.noop_driver package

Submodules

octavia.network.drivers.noop_driver.driver module

class NoopManager

Bases: `object`

allocate_vip(*loadbalancer*)

apply_qos_on_port(*qos_id, port_id*)

create_port(*network_id, name=None, fixed_ips=(), secondary_ips=(), security_group_ids=(), admin_state_up=True, qos_policy_id=None*)

deallocate_vip(*vip*)

delete_port(*port_id*)

failover_preparation(*amphora*)

get_network(*network_id*)

get_network_by_name(*network_name*)

get_network_configs(*loadbalancer, amphora=None*)

get_network_ip_availability(*network*)

get_plugged_networks(*compute_id*)

get_port(*port_id*)

get_port_by_name(*port_name*)

get_port_by_net_id_device_id(*network_id, device_id*)

get_qos_policy(*qos_policy_id*)

get_security_group(*sg_name*)

get_subnet(*subnet_id*)

get_subnet_by_name(*subnet_name*)

plug_aap_port(*load_balancer, vip, amphora, subnet*)

plug_fixed_ip(*port_id, subnet_id, ip_address=None*)

plug_network(*compute_id, network_id*)

plug_port(*amphora, port*)

plug_vip(*loadbalancer, vip*)

qos_enabled()

set_port_admin_state_up(*port_id*, *state*)

unplug_aap_port(*vip*, *amphora*, *subnet*)

unplug_fixed_ip(*port_id*, *subnet_id*)

unplug_network(*compute_id*, *network_id*)

unplug_vip(*loadbalancer*, *vip*)

update_vip(*loadbalancer*, *for_delete=False*)

update_vip_sg(*load_balancer*, *vip*)

wait_for_port_detach(*amphora*)

class NoopNetworkDriver

Bases: `octavia.network.base.AbstractNetworkDriver`

allocate_vip(*loadbalancer*)

Allocates a virtual ip.

Reserves it for later use as the frontend connection of a load balancer.

Parameters **load_balancer** -- octavia.common.data_models.LoadBalancer instance

Returns octavia.common.data_models.VIP

Raises AllocateVIPException, PortNotFound, SubnetNotFound

apply_qos_on_port(*qos_id*, *port_id*)

create_port(*network_id*, *name=None*, *fixed_ips=()*, *secondary_ips=()*, *security_group_ids=()*, *admin_state_up=True*, *qos_policy_id=None*)

Creates a network port.

fixed_ips = [{'subnet_id': <id>, ('ip_address': <IP>')},] ip_address is optional in the fixed_ips dictionary.

Parameters

- **network_id** -- The network the port should be created on.
- **name** -- The name to apply to the port.
- **fixed_ips** -- A list of fixed IP dicts.
- **secondary_ips** -- A list of secondary IPs to add to the port.
- **security_group_ids** -- A list of security group IDs for the port.
- **qos_policy_id** -- The QoS policy ID to apply to the port.

Returns port A port data model object.

deallocate_vip(*vip*)

Removes any resources that reserved this virtual ip.

Parameters **vip** -- octavia.common.data_models.VIP instance

Returns None

Raises DeallocateVIPException, VIPInUseException, VIPConfiigurationNotFound `delete_port`(*port_id*)

Delete a network port.

Parameters `port_id` -- The port ID to delete.

Returns None

`failover_preparation`(*amphora*)

Prepare an amphora for failover.

Parameters `amphora` -- amphora object to failover

Returns None

Raises PortNotFound

`get_network`(*network_id*, *context=None*)

Retrieves network from network id.

Parameters

- `network_id` -- id of an network to retrieve
- `context` -- A request context

Returns octavia.network.data_models.Network

Raises NetworkException, NetworkNotFound

`get_network_by_name`(*network_name*)

Retrieves network from network name.

Parameters `network_name` -- name of a network to retrieve

Returns octavia.network.data_models.Network

Raises NetworkException, NetworkNotFound

`get_network_configs`(*loadbalancer*, *amphora=None*)

Retrieve network configurations

This method assumes that a dictionary of AmphoraNetworkConfigs keyed off of the related amphora id are returned. The configs contain data pertaining to each amphora that is later used for finalization of the entire load balancer configuration. The data provided to these configs is left up to the driver, this means the driver is responsible for providing data that is appropriate for the amphora network configurations.

Example return: {<amphora.id>: <AmphoraNetworkConfig>}

Parameters

- `load_balancer` -- The load_balancer configuration
- `amphora` -- Optional amphora to only query.

Returns dict of octavia.network.data_models.AmphoraNetworkConfig keyed off of the amphora id the config is associated with.

Raises NotFound, NetworkNotFound, SubnetNotFound, PortNotFound

`get_network_ip_availability`(*network*)

Retrieves network IP availability.

Parameters `network` -- octavia.network.data_models.Network

Returns octavia.network.data_models.Network_IP_Availability

Raises NetworkException, NetworkNotFound

`get_plugged_networks`(*compute_id*)

Retrieves the current plugged networking configuration.

Parameters `compute_id` -- id of an amphora in the compute service

Returns [octavia.network.data_models.Instance]

`get_port`(*port_id*, *context=None*)

Retrieves port from port id.

Parameters

- `port_id` -- id of a port to retrieve
- `context` -- A request context

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`get_port_by_name`(*port_name*)

Retrieves port from port name.

Parameters `port_name` -- name of a port to retrieve

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`get_port_by_net_id_device_id`(*network_id*, *device_id*)

Retrieves port from network id and device id.

Parameters

- `network_id` -- id of a network to filter by
- `device_id` -- id of a network device to filter by

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`get_qos_policy`(*qos_policy_id*)

`get_security_group`(*sg_name*)

Retrieves the security group by it's name.

Parameters `sg_name` -- The security group name.

Returns octavia.network.data_models.SecurityGroup, None if not enabled

Raises NetworkException, SecurityGroupNotFound

`get_subnet`(*subnet_id*, *context=None*)

Retrieves subnet from subnet id.

Parameters

- `subnet_id` -- id of a subnet to retrieve
- `context` -- A request context

Returns octavia.network.data_models.Subnet

Raises NetworkException, SubnetNotFound

`get_subnet_by_name`(*subnet_name*)

Retrieves subnet from subnet name.

Parameters **`subnet_name`** -- name of a subnet to retrieve

Returns octavia.network.data_models.Subnet

Raises NetworkException, SubnetNotFound

`plug_aap_port`(*load_balancer*, *vip*, *amphora*, *subnet*)

Plugs the AAP port to the amp

Parameters

- **`load_balancer`** -- Load Balancer to prepare the VIP for
- **`vip`** -- The VIP to plug
- **`amphora`** -- The amphora to plug the VIP into
- **`subnet`** -- The subnet to plug the aap into

`plug_fixed_ip`(*port_id*, *subnet_id*, *ip_address=None*)

Plug a fixed ip to an existing port.

If ip_address is not specified, one will be auto-assigned.

Parameters

- **`port_id`** -- id of a port to add a fixed ip
- **`subnet_id`** -- id of a subnet
- **`ip_address`** -- specific ip_address to add

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`plug_network`(*compute_id*, *network_id*)

Connects an existing amphora to an existing network.

Parameters

- **`compute_id`** -- id of an amphora in the compute service
- **`network_id`** -- id of a network

Returns octavia.network.data_models.Interface instance

Raises PlugNetworkException, AmphoraNotFound, NetworkNotFound

`plug_port`(*amphora*, *port*)

Plug a neutron port in to a compute instance

Parameters

- **`amphora`** -- amphora object to plug the port into
- **`port`** -- port to plug into the compute instance

Returns None

Raises PlugNetworkException, AmphoraNotFound, NetworkNotFound

`plug_vip`(*loadbalancer*, *vip*)

Plugs a virtual ip as the frontend connection of a load balancer.

Sets up the routing of traffic from the vip to the load balancer and its amphorae.

Parameters

- **`load_balancer`** -- octavia.common.data_models.LoadBalancer instance
- **`vip`** -- octavia.common.data_models.VIP instance

Returns dict consisting of amphora_id as key and bind_ip as value. bind_ip is the ip that the amphora should listen on to receive traffic to load balance.

Raises PlugVIPException, PortNotFound

`qos_enabled`()

Whether QoS is enabled

Returns Boolean

`set_port_admin_state_up`(*port_id*, *state*)

Set the admin state of a port. True is up, False is down.

Parameters

- **`port_id`** -- The port ID to update.
- **`state`** -- True for up, False for down.

Returns None

`unplug_aap_port`(*vip*, *amphora*, *subnet*)

Unplugs the AAP port to the amp

Parameters

- **`vip`** -- The VIP to plug
- **`amphora`** -- The amphora to plug the VIP into
- **`subnet`** -- The subnet to plug the aap into

`unplug_fixed_ip`(*port_id*, *subnet_id*)

Unplug a fixed ip from an existing port.

Parameters

- **`port_id`** -- id of a port to remove the fixed ip from
- **`subnet_id`** -- id of a subnet

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`unplug_network`(*compute_id*, *network_id*)

Disconnects an existing amphora from an existing network.

If ip_address is not specified, all the interfaces plugged on network_id should be unplugged.

Parameters

- **`compute_id`** -- id of an amphora in the compute service

- **network_id** -- id of a network

Returns None

Raises UnplugNetworkException, AmphoraNotFound, NetworkNotFound, NetworkException

unplug_vip(*loadbalancer*, *vip*)

Unplugs a virtual ip as the frontend connection of a load balancer.

Removes the routing of traffic from the vip to the load balancer and its amphorae.

Parameters

- **load_balancer** -- octavia.common.data_models.LoadBalancer instance
- **vip** -- octavia.common.data_models.VIP instance

Returns octavia.common.data_models.VIP instance

Raises UnplugVIPException, PluggedVIPNotFound

update_vip(*loadbalancer*, *for_delete=False*)

Hook for the driver to update the VIP information.

This method will be called upon the change of a load_balancer configuration. It is an optional method to be implemented by drivers. It allows the driver to update any VIP information based on the state of the passed in load_balancer.

Parameters

- **load_balancer** -- octavia.common.data_models.LoadBalancer instance
- **for_delete** -- Boolean indicating if this update is for a delete

Raises MissingVIPSecurityGroup

Returns None

update_vip_sg(*load_balancer*, *vip*)

Updates the security group for a VIP

Parameters

- **load_balancer** -- Load Balancer to rpepare the VIP for
- **vip** -- The VIP to plug

wait_for_port_detach(*amphora*)

Waits for the amphora ports device_id to be unset.

This method waits for the ports on an amphora device_id parameter to be " or None which signifies that nova has finished detaching the port from the instance.

Parameters **amphora** -- Amphora to wait for ports to detach.

Returns None

Raises

- *TimeoutException* -- Port did not detach in interval.
- *PortNotFound* -- Port was not found by neutron.

Module contents

Module contents

Submodules

octavia.network.base module

class AbstractNetworkDriver

Bases: `object`

This class defines the methods for a fully functional network driver.

Implementations of this interface can expect a rollback to occur if any of the non-nullipotent methods raise an exception.

abstract allocate_vip(*load_balancer*)

Allocates a virtual ip.

Reserves it for later use as the frontend connection of a load balancer.

Parameters **load_balancer** -- octavia.common.data_models.LoadBalancer instance

Returns octavia.common.data_models.VIP

Raises AllocateVIPException, PortNotFound, SubnetNotFound

abstract create_port(*network_id*, *name=None*, *fixed_ips=()*, *secondary_ips=()*, *security_group_ids=()*, *admin_state_up=True*, *qos_policy_id=None*)

Creates a network port.

fixed_ips = [{'subnet_id': <id>, ('ip_address': <IP>')},] ip_address is optional in the fixed_ips dictionary.

Parameters

- **network_id** -- The network the port should be created on.
- **name** -- The name to apply to the port.
- **fixed_ips** -- A list of fixed IP dicts.
- **secondary_ips** -- A list of secondary IPs to add to the port.
- **security_group_ids** -- A list of security group IDs for the port.
- **qos_policy_id** -- The QoS policy ID to apply to the port.

Returns port A port data model object.

abstract deallocate_vip(*vip*)

Removes any resources that reserved this virtual ip.

Parameters **vip** -- octavia.common.data_models.VIP instance

Returns None

Raises DeallocateVIPException, VIPInUseException, VIPConfiigurationNotFound abstract **delete_port**(*port_id*)

Delete a network port.

Parameters `port_id` -- The port ID to delete.

Returns None

abstract **failover_preparation**(*amphora*)

Prepare an amphora for failover.

Parameters `amphora` -- amphora object to failover

Returns None

Raises PortNotFound

abstract **get_network**(*network_id*, *context=None*)

Retrieves network from network id.

Parameters

- `network_id` -- id of an network to retrieve
- `context` -- A request context

Returns octavia.network.data_models.Network

Raises NetworkException, NetworkNotFound

abstract **get_network_by_name**(*network_name*)

Retrieves network from network name.

Parameters `network_name` -- name of a network to retrieve

Returns octavia.network.data_models.Network

Raises NetworkException, NetworkNotFound

abstract **get_network_configs**(*load_balancer*, *amphora=None*)

Retrieve network configurations

This method assumes that a dictionary of AmphoraNetworkConfigs keyed off of the related amphora id are returned. The configs contain data pertaining to each amphora that is later used for finalization of the entire load balancer configuration. The data provided to these configs is left up to the driver, this means the driver is responsible for providing data that is appropriate for the amphora network configurations.

Example return: {<amphora.id>: <AmphoraNetworkConfig>}

Parameters

- `load_balancer` -- The load_balancer configuration
- `amphora` -- Optional amphora to only query.

Returns dict of octavia.network.data_models.AmphoraNetworkConfig keyed off of the amphora id the config is associated with.

Raises NotFound, NetworkNotFound, SubnetNotFound, PortNotFound

abstract **get_network_ip_availability**(*network*)

Retrieves network IP availability.

Parameters `network` -- octavia.network.data_models.Network

Returns octavia.network.data_models.Network_IP_Availability

Raises NetworkException, NetworkNotFound

abstract get_plugged_networks(*compute_id*)

Retrieves the current plugged networking configuration.

Parameters **compute_id** -- id of an amphora in the compute service

Returns [octavia.network.data_models.Instance]

abstract get_port(*port_id*, *context=None*)

Retrieves port from port id.

Parameters

- **port_id** -- id of a port to retrieve
- **context** -- A request context

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

abstract get_port_by_name(*port_name*)

Retrieves port from port name.

Parameters **port_name** -- name of a port to retrieve

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

abstract get_port_by_net_id_device_id(*network_id*, *device_id*)

Retrieves port from network id and device id.

Parameters

- **network_id** -- id of a network to filter by
- **device_id** -- id of a network device to filter by

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

abstract get_security_group(*sg_name*)

Retrieves the security group by it's name.

Parameters **sg_name** -- The security group name.

Returns octavia.network.data_models.SecurityGroup, None if not enabled

Raises NetworkException, SecurityGroupNotFound

abstract get_subnet(*subnet_id*, *context=None*)

Retrieves subnet from subnet id.

Parameters

- **subnet_id** -- id of a subnet to retrieve
- **context** -- A request context

Returns octavia.network.data_models.Subnet

Raises NetworkException, SubnetNotFound

`abstract get_subnet_by_name`(*subnet_name*)
Retrieves subnet from subnet name.

Parameters **`subnet_name`** -- name of a subnet to retrieve

Returns octavia.network.data_models.Subnet

Raises NetworkException, SubnetNotFound

`abstract plug_aap_port`(*load_balancer*, *vip*, *amphora*, *subnet*)
Plugs the AAP port to the amp

Parameters

- **`load_balancer`** -- Load Balancer to prepare the VIP for
- **`vip`** -- The VIP to plug
- **`amphora`** -- The amphora to plug the VIP into
- **`subnet`** -- The subnet to plug the aap into

`abstract plug_fixed_ip`(*port_id*, *subnet_id*, *ip_address=None*)
Plug a fixed ip to an existing port.

If ip_address is not specified, one will be auto-assigned.

Parameters

- **`port_id`** -- id of a port to add a fixed ip
- **`subnet_id`** -- id of a subnet
- **`ip_address`** -- specific ip_address to add

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

`abstract plug_network`(*compute_id*, *network_id*)
Connects an existing amphora to an existing network.

Parameters

- **`compute_id`** -- id of an amphora in the compute service
- **`network_id`** -- id of a network

Returns octavia.network.data_models.Interface instance

Raises PlugNetworkException, AmphoraNotFound, NetworkNotFound

`abstract plug_port`(*amphora*, *port*)
Plug a neutron port in to a compute instance

Parameters

- **`amphora`** -- amphora object to plug the port into
- **`port`** -- port to plug into the compute instance

Returns None

Raises PlugNetworkException, AmphoraNotFound, NetworkNotFound

abstract plug_vip(*load_balancer*, *vip*)

Plugs a virtual ip as the frontend connection of a load balancer.

Sets up the routing of traffic from the vip to the load balancer and its amphorae.

Parameters

- **load_balancer** -- octavia.common.data_models.LoadBalancer instance
- **vip** -- octavia.common.data_models.VIP instance

Returns dict consisting of amphora_id as key and bind_ip as value. bind_ip is the ip that the amphora should listen on to receive traffic to load balance.

Raises PlugVIPException, PortNotFound

abstract qos_enabled()

Whether QoS is enabled

Returns Boolean

abstract set_port_admin_state_up(*port_id*, *state*)

Set the admin state of a port. True is up, False is down.

Parameters

- **port_id** -- The port ID to update.
- **state** -- True for up, False for down.

Returns None

abstract unplug_aap_port(*vip*, *amphora*, *subnet*)

Unplugs the AAP port to the amp

Parameters

- **vip** -- The VIP to plug
- **amphora** -- The amphora to plug the VIP into
- **subnet** -- The subnet to plug the aap into

abstract unplug_fixed_ip(*port_id*, *subnet_id*)

Unplug a fixed ip from an existing port.

Parameters

- **port_id** -- id of a port to remove the fixed ip from
- **subnet_id** -- id of a subnet

Returns octavia.network.data_models.Port

Raises NetworkException, PortNotFound

abstract unplug_network(*compute_id*, *network_id*)

Disconnects an existing amphora from an existing network.

If ip_address is not specified, all the interfaces plugged on network_id should be unplugged.

Parameters

- **compute_id** -- id of an amphora in the compute service
- **network_id** -- id of a network

Returns None

Raises UnplugNetworkException, AmphoraNotFound, NetworkNotFound, NetworkException

`abstract unplug_vip`(*load_balancer*, *vip*)

Unplugs a virtual ip as the frontend connection of a load balancer.

Removes the routing of traffic from the vip to the load balancer and its amphorae.

Parameters

- **`load_balancer`** -- octavia.common.data_models.LoadBalancer instance
- **`vip`** -- octavia.common.data_models.VIP instance

Returns octavia.common.data_models.VIP instance

Raises UnplugVIPException, PluggedVIPNotFound

`update_vip`(*load_balancer*, *for_delete*)

Hook for the driver to update the VIP information.

This method will be called upon the change of a load_balancer configuration. It is an optional method to be implemented by drivers. It allows the driver to update any VIP information based on the state of the passed in load_balancer.

Parameters

- **`load_balancer`** -- octavia.common.data_models.LoadBalancer instance
- **`for_delete`** -- Boolean indicating if this update is for a delete

Raises MissingVIPSecurityGroup

Returns None

`abstract update_vip_sg`(*load_balancer*, *vip*)

Updates the security group for a VIP

Parameters

- **`load_balancer`** -- Load Balancer to rpepare the VIP for
- **`vip`** -- The VIP to plug

`abstract wait_for_port_detach`(*amphora*)

Waits for the amphora ports device_id to be unset.

This method waits for the ports on an amphora device_id parameter to be ” or None which signifies that nova has finished detaching the port from the instance.

Parameters **`amphora`** -- Amphora to wait for ports to detach.

Returns None

Raises

- *`TimeoutException`* -- Port did not detach in interval.
- *`PortNotFound`* -- Port was not found by neutron.

`exception AllocateVIPException`(**args*, ***kwargs*)

Bases: *`octavia.network.base.NetworkException`*

exception AmphoraNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception CreatePortException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception DeallocateVIPException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception NetworkException(**args*, ***kwargs*)
Bases: *octavia.common.exceptions.OctaviaException*

exception NetworkNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception PlugNetworkException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception PlugVIPException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception PluggedVIPNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception PortNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception QosPolicyNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception SecurityGroupNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception SubnetNotFound(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception TimeoutException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception UnplugNetworkException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception UnplugVIPException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

exception VIPInUseException(**args*, ***kwargs*)
Bases: *octavia.network.base.NetworkException*

octavia.network.data_models module

class AmphoraNetworkConfig(*amphora=None*, *vip_subnet=None*, *vip_port=None*, *vrrp_subnet=None*, *vrrp_port=None*, *ha_subnet=None*, *ha_port=None*)
Bases: `octavia.common.data_models.`

class Delta(*amphora_id=None*, *compute_id=None*, *add_nics=None*, *delete_nics=None*, *add_subnets=None*, *delete_subnets=None*)
Bases: `octavia.common.data_models.`

class FixedIP(*subnet_id=None*, *ip_address=None*, *subnet=None*)
Bases: `octavia.common.data_models.`

class FloatingIP(*id=None*, *description=None*, *project_id=None*, *status=None*, *router_id=None*, *port_id=None*, *floating_network_id=None*, *floating_ip_address=None*, *fixed_ip_address=None*, *fixed_port_id=None*)
Bases: `octavia.common.data_models.`

class HostRoute(*nexthop=None*, *destination=None*)
Bases: `octavia.common.data_models.`

class Interface(*id=None*, *compute_id=None*, *network_id=None*, *fixed_ips=None*, *port_id=None*)
Bases: `octavia.common.data_models.`

class Network(*id=None*, *name=None*, *subnets=None*, *project_id=None*, *admin_state_up=None*, *mtu=None*, *provider_network_type=None*, *provider_physical_network=None*, *provider_segmentation_id=None*, *router_external=None*, *port_security_enabled=None*)
Bases: `octavia.common.data_models.`

class Network_IP_Availability(*network_id=None*, *tenant_id=None*, *project_id=None*, *network_name=None*, *total_ips=None*, *used_ips=None*, *subnet_ip_availability=None*)
Bases: `octavia.common.data_models.`

class Port(*id=None*, *name=None*, *device_id=None*, *device_owner=None*, *mac_address=None*, *network_id=None*, *status=None*, *project_id=None*, *admin_state_up=None*, *fixed_ips=None*, *network=None*, *qos_policy_id=None*, *security_group_ids=None*)
Bases: `octavia.common.data_models.`

get_subnet_id(*fixed_ip_address*)

class QosPolicy(*id*)
Bases: `octavia.common.data_models.`

class SecurityGroup(*id=None*, *project_id=None*, *name=None*, *description=None*, *security_group_rule_ids=None*, *tags=None*, *stateful=None*)
Bases: `octavia.common.data_models.`

class Subnet(*id=None*, *name=None*, *network_id=None*, *project_id=None*, *gateway_ip=None*, *cidr=None*, *ip_version=None*, *host_routes=None*)
Bases: `octavia.common.data_models.`

Module contents

octavia.policies package

Submodules

octavia.policies.amphora module

list_rules()

octavia.policies.availability_zone module

`list_rules()`

octavia.policies.availability_zone_profile module

`list_rules()`

octavia.policies.base module

`list_rules()`

octavia.policies.flavor module

`list_rules()`

octavia.policies.flavor_profile module

`list_rules()`

octavia.policies.healthmonitor module

`list_rules()`

octavia.policies.l7policy module

`list_rules()`

octavia.policies.l7rule module

`list_rules()`

octavia.policies.listener module

`list_rules()`

octavia.policies.loadbalancer module

`list_rules()`

octavia.policies.member module

`list_rules()`

octavia.policies.pool module

`list_rules()`

octavia.policies.provider module

`list_rules()`

octavia.policies.provider_availability_zone module

`list_rules()`

octavia.policies.provider_flavor module

`list_rules()`

octavia.policies.quota module

`list_rules()`

Module contents

`list_rules()`

octavia.statistics package

Subpackages

octavia.statistics.drivers package

Submodules

octavia.statistics.drivers.logger module

class StatsLogger

Bases: *octavia.statistics.stats_base.StatsDriverMixin*

update_stats(*listener_stats*, *deltas=False*)

Return a stats object formatted for a generic backend

Parameters

- **listener_stats** (*list*) -- A list of data_model.ListenerStatistics objects
- **deltas** (*bool*) -- Indicates whether the stats are deltas (false==absolute)

octavia.statistics.drivers.update_db module

class StatsUpdateDb

Bases: *octavia.statistics.stats_base.StatsDriverMixin*

update_stats(*listener_stats*, *deltas=False*)

This function is to update the db with listener stats

Module contents

Submodules

octavia.statistics.stats_base module

class StatsDriverMixin

Bases: `object`

abstract update_stats(*listener_stats*, *deltas=False*)

Return a stats object formatted for a generic backend

Parameters

- **listener_stats** (*list*) -- A list of data_model.ListenerStatistics objects
- **deltas** (*bool*) -- Indicates whether the stats are deltas (false==absolute)

update_stats_via_driver(*listener_stats*, *deltas=False*)

Send listener stats to the enabled stats driver(s)

Parameters

- **listener_stats** (*list*) -- A list of ListenerStatistics objects
- **deltas** (*bool*) -- Indicates whether the stats are deltas (false==absolute)

Module contents

octavia.volume package

Subpackages

octavia.volume.drivers package

Subpackages

octavia.volume.drivers.noop_driver package

Submodules

octavia.volume.drivers.noop_driver.driver module

class NoopManager

Bases: `object`

create_volume_from_image(*image_id*)

delete_volume(*volume_id*)

get_image_from_volume(*volume_id*)

class NoopVolumeDriver

Bases: *octavia.volume.volume_base.VolumeBase*

create_volume_from_image(*image_id*)

Create volume for instance

Parameters **image_id** -- ID of amphora image

:return volume id

delete_volume(*volume_id*)

Delete volume

Parameters **volume_id** -- ID of amphora volume

get_image_from_volume(*volume_id*)

Get cinder volume

Parameters **volume_id** -- ID of amphora volume

:return image id

Module contents

Submodules

octavia.volume.drivers.cinder_driver module

class VolumeManager

Bases: *octavia.volume.volume_base.VolumeBase*

Volume implementation of virtual machines via cinder.

create_volume_from_image(*image_id*)

Create cinder volume

Parameters **image_id** -- ID of amphora image

:return volume id

delete_volume(*volume_id*)

Get glance image from volume

Parameters **volume_id** -- ID of amphora boot volume

:return image id

get_image_from_volume(*volume_id*)

Get glance image from volume

Parameters **volume_id** -- ID of amphora boot volume

:return image id

Module contents

Submodules

octavia.volume.volume_base module

class VolumeBase

Bases: `object`

abstract create_volume_from_image(*image_id*)

Create volume for instance

Parameters **image_id** -- ID of amphora image

:return volume id

abstract delete_volume(*volume_id*)

Delete volume

Parameters **volume_id** -- ID of amphora volume

abstract get_image_from_volume(*volume_id*)

Get cinder volume

Parameters **volume_id** -- ID of amphora volume

:return image id

Module contents

Submodules

octavia.i18n module

octavia.opts module

`add_auth_opts()`

`list_opts()`

octavia.version module

`product_string()`

`vendor_string()`

`version_string_with_package()`

Module contents

CHAPTER

FIVE

OCTAVIA INSTALLATION

5.1 Install and configure

This section describes how to install and configure the Load-balancer service, code-named Octavia, on the controller node.

This section assumes that you already have a working OpenStack environment with at least the following components installed: Identity Service, Image Service, Placement Service, Compute Service, and Networking Service.

Note that installation and configuration vary by distribution.

5.1.1 Install and configure for Ubuntu

This section describes how to install and configure the Load-balancer service for Ubuntu 18.04 (LTS).

Prerequisites

Before you install and configure the service, you must create a database, service credentials, and API endpoints.

1. Create the database, complete these steps:

   - Use the database access client to connect to the database server as the `root` user:

     ```
     # mysql
     ```

   - Create the `octavia` database:

     ```
     CREATE DATABASE octavia;
     ```

   - Grant proper access to the `octavia` database:

     ```
     GRANT ALL PRIVILEGES ON octavia.* TO 'octavia'@'localhost' \
     IDENTIFIED BY 'OCTAVIA_DBPASS';
     GRANT ALL PRIVILEGES ON octavia.* TO 'octavia'@'%' \
     IDENTIFIED BY 'OCTAVIA_DBPASS';
     ```

     Replace OCTAVIA_DBPASS with a suitable password.

   - Exit the database access client.

```
exit;
```

2. Source the `admin` credentials to gain access to admin-only CLI commands:

```
$ . admin-openrc
```

3. To create the Octavia service credentials, complete these steps:

   - Create the `octavia` user:

```
$ openstack user create --domain default --password-prompt octavia
  User Password:
  Repeat User Password:
  +---------------------+----------------------------------+
  | Field               | Value                            |
  +---------------------+----------------------------------+
  | domain_id           | default                          |
  | enabled             | True                             |
  | id                  | b18ee38e06034b748141beda8fc8bfad |
  | name                | octavia                          |
  | options             | {}                               |
  | password_expires_at | None                             |
  +---------------------+----------------------------------+
```

   - Add the `admin` role to the `octavia` user:

```
$ openstack role add --project service --user octavia admin
```

Note: This command produces no output.

Note: The Octavia service does not require the full admin role. Details of how to run Octavia without the admin role will come in a future version of this document.

   - Create the octavia service entities:

```
$ openstack service create --name octavia --description "OpenStack␣
↪Octavia" load-balancer
  +-------------+----------------------------------+
  | Field       | Value                            |
  +-------------+----------------------------------+
  | description | OpenStack Octavia                |
  | enabled     | True                             |
  | id          | d854f6fff0a64f77bda8003c8dedfada |
  | name        | octavia                          |
  | type        | load-balancer                    |
  +-------------+----------------------------------+
```

4. Create the Load-balancer service API endpoints:

```
$ openstack endpoint create --region RegionOne \
  load-balancer public http://controller:9876
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 47cf883de46242c39f147c52f2958ebf |
  | interface    | public                           |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | d854f6fff0a64f77bda8003c8dedfada |
  | service_name | octavia                          |
  | service_type | load-balancer                    |
  | url          | http://controller:9876           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  load-balancer internal http://controller:9876
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 225aef8465ef4df48a341aaaf2b0a390 |
  | interface    | internal                         |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | d854f6fff0a64f77bda8003c8dedfada |
  | service_name | octavia                          |
  | service_type | load-balancer                    |
  | url          | http://controller:9876           |
  +--------------+----------------------------------+

$ openstack endpoint create --region RegionOne \
  load-balancer admin http://controller:9876
  +--------------+----------------------------------+
  | Field        | Value                            |
  +--------------+----------------------------------+
  | enabled      | True                             |
  | id           | 375eb5057fb546edbdf3ee4866179672 |
  | interface    | admin                            |
  | region       | RegionOne                        |
  | region_id    | RegionOne                        |
  | service_id   | d854f6fff0a64f77bda8003c8dedfada |
  | service_name | octavia                          |
  | service_type | load-balancer                    |
  | url          | http://controller:9876           |
  +--------------+----------------------------------+
```

5. Create octavia-openrc file

```
cat << EOF >> $HOME/octavia-openrc
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=service
export OS_USERNAME=octavia
export OS_PASSWORD=OCTAVIA_PASS
export OS_AUTH_URL=http://controller:5000
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
export OS_VOLUME_API_VERSION=3
EOF
```

Replace OCTAVIA_PASS with the password you chose for the octavia user in the Identity service.

6. Source the `octavia` credentials to gain access to octavia CLI commands:

```
$ . $HOME/octavia-openrc
```

7. Create the amphora image

   For creating amphora image, please refer to the Building Octavia Amphora Images.

8. Upload the amphora image

```
$ openstack image create --disk-format qcow2 --container-format bare \
  --private --tag amphora \
  --file <path to the amphora image> amphora-x64-haproxy
```

9. Create a flavor for the amphora image

```
$ openstack flavor create --id 200 --vcpus 1 --ram 1024 \
  --disk 2 "amphora" --private
```

Install and configure components

1. Install the packages:

```
# apt install octavia-api octavia-health-manager octavia-housekeeping \
  octavia-worker python3-octavia python3-octaviaclient
```

   If octavia-common and octavia-api packages ask you to configure, choose No.

2. Create the certificates

```
$ git clone https://opendev.org/openstack/octavia.git
$ cd octavia/bin/
$ source create_dual_intermediate_CA.sh
$ sudo mkdir -p /etc/octavia/certs/private
$ sudo chmod 755 /etc/octavia -R
$ sudo cp -p etc/octavia/certs/server_ca.cert.pem /etc/octavia/certs
$ sudo cp -p etc/octavia/certs/server_ca-chain.cert.pem /etc/octavia/certs
$ sudo cp -p etc/octavia/certs/server_ca.key.pem /etc/octavia/certs/
↪private
```

(continues on next page)

(continued from previous page)

```
$ sudo cp -p etc/octavia/certs/client_ca.cert.pem /etc/octavia/certs
$ sudo cp -p etc/octavia/certs/client.cert-and-key.pem /etc/octavia/certs/
↪private
```

For the production environment, Please refer to the Octavia Certificate Configuration Guide.

3. Source the `octavia` credentials to gain access to octavia CLI commands:

```
$ . octavia-openrc
```

4. Create security groups and their rules

```
$ openstack security group create lb-mgmt-sec-grp
$ openstack security group rule create --protocol icmp lb-mgmt-sec-grp
$ openstack security group rule create --protocol tcp --dst-port 22 lb-
↪mgmt-sec-grp
$ openstack security group rule create --protocol tcp --dst-port 9443 lb-
↪mgmt-sec-grp
$ openstack security group create lb-health-mgr-sec-grp
$ openstack security group rule create --protocol udp --dst-port 5555 lb-
↪health-mgr-sec-grp
```

5. Create a key pair for logging in to the amphora instance

```
$ openstack keypair create --public-key ~/.ssh/id_rsa.pub mykey
```

Note: Check whether " ~/.ssh/id_rsa.pub" file exists or not in advance. If the file does not exist, run the ssh-keygen command to create it.

6. Create dhclient.conf file for dhclient

```
$ cd $HOME
$ sudo mkdir -m755 -p /etc/dhcp/octavia
$ sudo cp octavia/etc/dhcp/dhclient.conf /etc/dhcp/octavia
```

7. Create a network

Note: During the execution of the below command, please save the of BRNAME and MGMT_PORT_MAC in a notepad for further reference.

```
$ OCTAVIA_MGMT_SUBNET=172.16.0.0/12
$ OCTAVIA_MGMT_SUBNET_START=172.16.0.100
$ OCTAVIA_MGMT_SUBNET_END=172.16.31.254
$ OCTAVIA_MGMT_PORT_IP=172.16.0.2

$ openstack network create lb-mgmt-net
$ openstack subnet create --subnet-range $OCTAVIA_MGMT_SUBNET --
↪allocation-pool \
```

(continues on next page)

(continued from previous page)

```
  start=$OCTAVIA_MGMT_SUBNET_START,end=$OCTAVIA_MGMT_SUBNET_END \
  --network lb-mgmt-net lb-mgmt-subnet

$ SUBNET_ID=$(openstack subnet show lb-mgmt-subnet -f value -c id)
$ PORT_FIXED_IP="--fixed-ip subnet=$SUBNET_ID,ip-address=$OCTAVIA_MGMT_
↪PORT_IP"

$ MGMT_PORT_ID=$(openstack port create --security-group \
  lb-health-mgr-sec-grp --device-owner Octavia:health-mgr \
  --host=$(hostname) -c id -f value --network lb-mgmt-net \
  $PORT_FIXED_IP octavia-health-manager-listen-port)

$ MGMT_PORT_MAC=$(openstack port show -c mac_address -f value \
  $MGMT_PORT_ID)

$ sudo ip link add o-hm0 type veth peer name o-bhm0
$ NETID=$(openstack network show lb-mgmt-net -c id -f value)
$ BRNAME=brq$(echo $NETID|cut -c 1-11)
$ sudo brctl addif $BRNAME o-bhm0
$ sudo ip link set o-bhm0 up

$ sudo ip link set dev o-hm0 address $MGMT_PORT_MAC
$ sudo iptables -I INPUT -i o-hm0 -p udp --dport 5555 -j ACCEPT
$ sudo dhclient -v o-hm0 -cf /etc/dhcp/octavia
```

8. Below settings are required to create veth pair after the host reboot

   Edit the `/etc/systemd/network/o-hm0.network` file

```
[Match]
Name=o-hm0

[Network]
DHCP=yes
```

   Edit the `/etc/systemd/system/octavia-interface.service` file

```
[Unit]
Description=Octavia Interface Creator
Requires=neutron-linuxbridge-agent.service
After=neutron-linuxbridge-agent.service

[Service]
Type=oneshot
RemainAfterExit=true
ExecStart=/opt/octavia-interface.sh start
ExecStop=/opt/octavia-interface.sh stop

[Install]
WantedBy=multi-user.target
```

Edit the `/opt/octavia-interface.sh` file

```
#!/bin/bash

set -ex

MAC=$MGMT_PORT_MAC
BRNAME=$BRNAME

if [ "$1" == "start" ]; then
  ip link add o-hm0 type veth peer name o-bhm0
  brctl addif $BRNAME o-bhm0
  ip link set o-bhm0 up
  ip link set dev o-hm0 address $MAC
  ip link set o-hm0 up
  iptables -I INPUT -i o-hm0 -p udp --dport 5555 -j ACCEPT
elif [ "$1" == "stop" ]; then
  ip link del o-hm0
else
  brctl show $BRNAME
  ip a s dev o-hm0
fi
```

You need to substitute $MGMT_PORT_MAC and $BRNAME for the values in your environment.

9. Edit the `/etc/octavia/octavia.conf` file

   - In the `[database]` section, configure database access:

     ```
     [database]
     connection = mysql+pymysql://octavia:OCTAVIA_DBPASS@controller/
     ↪octavia
     ```

     Replace OCTAVIA_DBPASS with the password you chose for the Octavia databases.

   - In the `[DEFAULT]` section, configure the transport url for RabbitMQ message broker.

     ```
     [DEFAULT]
     transport_url = rabbit://openstack:RABBIT_PASS@controller
     ```

     Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

   - In the `[oslo_messaging]` section, configure the transport url for RabbitMQ message broker and topic name.

     ```
     [oslo_messaging]
     ...
     topic = octavia_prov
     ```

     Replace RABBIT_PASS with the password you chose for the openstack account in RabbitMQ.

   - In the `[api_settings]` section, configure the host IP and port to bind to.

```
[api_settings]
bind_host = 0.0.0.0
bind_port = 9876
```

- In the `[keystone_authtoken]` section, configure Identity service access.

```
[keystone_authtoken]
www_authenticate_uri = http://controller:5000
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = octavia
password = OCTAVIA_PASS
```

  Replace OCTAVIA_PASS with the password you chose for the octavia user in the Identity service.

- In the `[service_auth]` section, configure credentials for using other openstack services

```
[service_auth]
auth_url = http://controller:5000
memcached_servers = controller:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = octavia
password = OCTAVIA_PASS
```

  Replace OCTAVIA_PASS with the password you chose for the octavia user in the Identity service.

- In the `[certificates]` section, configure the absolute path to the CA Certificate, the Private Key for signing, and passphrases.

```
[certificates]
...
server_certs_key_passphrase = insecure-key-do-not-use-this-key
ca_private_key_passphrase = not-secure-passphrase
ca_private_key = /etc/octavia/certs/private/server_ca.key.pem
ca_certificate = /etc/octavia/certs/server_ca.cert.pem
```

  Note: The values of ca_private_key_passphrase and server_certs_key_passphrase are default and should not be used in production. The server_certs_key_passphrase must be a base64 compatible and 32 characters long string.

- In the `[haproxy_amphora]` section, configure the client certificate and the CA.

```
[haproxy_amphora]
...
server_ca = /etc/octavia/certs/server_ca-chain.cert.pem
client_cert = /etc/octavia/certs/private/client.cert-and-key.pem
```

- In the `[health_manager]` section, configure the IP and port number for heartbeat.

```
[health_manager]
...
bind_port = 5555
bind_ip = 172.16.0.2
controller_ip_port_list = 172.16.0.2:5555
```

- In the `[controller_worker]` section, configure worker settings.

```
[controller_worker]
...
amp_image_owner_id = <id of service project>
amp_image_tag = amphora
amp_ssh_key_name = mykey
amp_secgroup_list = <lb-mgmt-sec-grp_id>
amp_boot_network_list = <lb-mgmt-net_id>
amp_flavor_id = 200
network_driver = allowed_address_pairs_driver
compute_driver = compute_nova_driver
amphora_driver = amphora_haproxy_rest_driver
client_ca = /etc/octavia/certs/client_ca.cert.pem
```

10. Populate the octavia database:

```
# octavia-db-manage --config-file /etc/octavia/octavia.conf upgrade␣
↪head
```

Finalize installation

Restart the services:

```
# systemctl restart octavia-api octavia-health-manager octavia-
↪housekeeping octavia-worker
```

5.1.2 Additional configuration steps to configure amphorav2 provider

The amphorav2 provider driver improves control plane resiliency. Should a control plane host go down during a load balancer provisioning operation, an alternate controller can resume the in-process provisioning and complete the request. This solves the issue with resources stuck in PENDING_* states by writing info about task states in persistent backend and monitoring job claims via jobboard.

If you would like to use amphorav2 provider with jobboard-based controller for load-balancer service the following additional steps are required.

This provider driver can also run without jobboard and its dependencies (extra database, Redis/Zookeeper). This is the default setting while jobboard remains an experimental feature.

Prerequisites

Amphorav2 provider requires creation of additional database `octavia_persistence` to store info about state of tasks and progress of its execution. Also to monitor progress on taskflow jobs amphorav2 provider uses jobboard. As jobboard backend could be used Redis or Zookeeper key-value storages. Operator should chose the one that is more preferable for specific cloud. The default is Redis. Key-values storage clients should be install with extras [zookeeper] or [redis] during installation of octavia packages.

1. Create the database, complete these steps:
   - Use the database access client to connect to the database server as the `root` user:

     ```
     # mysql
     ```

   - Create the `octavia_persistence` database:

     ```
     CREATE DATABASE octavia_persistence;
     ```

   - Grant proper access to the `octavia_persistence` database:

     ```
     GRANT ALL PRIVILEGES ON octavia.* TO 'octavia_persistence'@'localhost
     ↪' \
     IDENTIFIED BY 'OCTAVIA_DBPASS';
     GRANT ALL PRIVILEGES ON octavia.* TO 'octavia_persistence'@'%' \
     IDENTIFIED BY 'OCTAVIA_DBPASS';
     ```

     Replace OCTAVIA_DBPASS with a suitable password.

2. Install desired key-value backend (Redis or Zookeper).

Additional configuration to octavia components

1. Edit the `/etc/octavia/octavia.conf` file `[task_flow]` section
   - Configure database access for persistence backend:

```
  [task_flow]
  persistence_connection = mysql+pymysql://octavia:OCTAVIA_
↪DBPASS@controller/octavia_persistence

Replace OCTAVIA_DBPASS with the password you chose for the␣
↪Octavia databases.
```

   - Set desired jobboard backend and its configuration:

```
[task_flow]
jobboard_enabled = True
jobboard_backend_driver = 'redis_taskflow_driver'
jobboard_backend_hosts = KEYVALUE_HOST_IPS
```

(continues on next page)

(continued from previous page)

```
jobboard_backend_port = KEYVALUE_PORT
jobboard_backend_password = OCTAVIA_JOBBOARDPASS
jobboard_backend_namespace = 'octavia_jobboard'
```

Replace OCTAVIA_JOBBOARDPASS with the password you chose for the Octavia key-value storage. Replace KEYVALUE_HOST_IPS and KEYVALUE_PORT with ip and port which chosen key-value storage is using.

2. Populate the octavia database:

```
# octavia-db-manage --config-file /etc/octavia/octavia.conf upgrade_
↪persistence
```

CHAPTER

SIX

OCTAVIA REFERENCE

CHAPTER

SEVEN

OCTAVIA USER

7.1 Cookbooks

7.1.1 Basic Load Balancing Cookbook

Introduction

This document contains several examples of using basic load balancing services as a tenant or "regular" cloud user.

For the purposes of this guide we assume that the neutron and barbican command-line interfaces, via the OpenStack client, are going to be used to configure all features of Octavia. In order to keep these examples short, we also assume that tasks not directly associated with deploying load balancing services have already been accomplished. This might include such things as deploying and configuring web servers, setting up Neutron networks, obtaining TLS certificates from a trusted provider, and so on. A description of the starting conditions is given in each example below.

Please also note that this guide assumes you are familiar with the specific load balancer terminology defined in the *Octavia Glossary*. For a description of load balancing itself and the Octavia project, please see: *Introducing Octavia*.

Examples

Deploy a basic HTTP load balancer

While this is technically the simplest complete load balancing solution that can be deployed, we recommend deploying HTTP load balancers with a health monitor to ensure back-end member availability. See *Deploy a basic HTTP load balancer with a health monitor* below.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an HTTP application on TCP port 80.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes web requests to the back-end servers.

Solution:

1. Create load balancer *lb1* on subnet *public-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool.
4. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
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
```

Deploy a basic HTTP load balancer with a health monitor

This is the simplest recommended load balancing solution for HTTP applications. This solution is appropriate for operators with provider networks that are not compatible with Neutron floating-ip functionality (such as IPv6 networks). However, if you need to retain control of the external IP through which a load balancer is accessible, even if the load balancer needs to be destroyed or recreated, it may be more appropriate to deploy your basic load balancer using a floating IP. See *Deploy a basic HTTP load balancer using a floating IP* below.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* 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 "/healthcheck". See *Configuration arguments for HTTP health monitors* below.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes web requests to the back-end servers, and which checks the "/healthcheck" path to ensure back-end member health.

Solution:

1. Create load balancer *lb1* on subnet *public-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool.
4. Create a health monitor on *pool1* which tests the "/healthcheck" path.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --
↪timeout 10 --type HTTP --url-path /healthcheck pool1
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
```

Deploy a basic HTTP load balancer using a floating IP

It can be beneficial to use a floating IP when setting up a load balancer's VIP in order to ensure you retain control of the IP that gets assigned as the floating IP in case the load balancer needs to be destroyed, moved, or recreated.

Note that this is not possible to do with IPv6 load balancers as floating IPs do not work with IPv6.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* 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 "/healthcheck". See *Configuration arguments for HTTP health monitors* below.
- Neutron network *public* is a shared external network created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes web requests to the back-end servers, and which checks the "/healthcheck" path to ensure back-end member health. Further, we want to do this using a floating IP.

Solution:

1. Create load balancer *lb1* on subnet *private-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool.
4. Create a health monitor on *pool1* which tests the "/healthcheck" path.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.
6. Create a floating IP address on *public-subnet*.
7. Associate this floating IP with the *lb1*'s VIP port.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id private-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --
↪timeout 10 --type HTTP --url-path /healthcheck pool1
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
openstack floating ip create public
# The following IDs should be visible in the output of previous commands
openstack floating ip set --port <load_balancer_vip_port_id> <floating_ip_id>
```

Deploy a basic HTTP load balancer with session persistence

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an HTTP application on TCP port 80.
- The application is written such that web clients should always be directed to the same back-end server throughout their web session, based on an application cookie inserted by the web application named 'PHPSESSIONID'.
- These back-end servers have been configured with a health check at the URL path "/healthcheck". See *Configuration arguments for HTTP health monitors* below.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes web requests to the back-end servers, persists sessions using the PHPSESSIONID as a key, and which checks the "/healthcheck" path to ensure back-end member health.

Solution:

1. Create load balancer *lb1* on subnet *public-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool which defines session persistence on the 'PHPSESSIONID' cookie.
4. Create a health monitor on *pool1* which tests the "/healthcheck" path.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
```

(continues on next page)

(continued from previous page)

```
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP --session-persistence type=APP_COOKIE,
↪cookie_name=PHPSESSIONID
openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --
↪timeout 10 --type HTTP --url-path /healthcheck pool1
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
```

Deploy a TCP load balancer

This is generally suitable when load balancing a non-HTTP TCP-based service.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an custom application on TCP port 23456
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes requests to the back-end servers.
- We want to employ a TCP health check to ensure that the back-end servers are available.

Solution:

1. Create load balancer *lb1* on subnet *public-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool.
4. Create a health monitor on *pool1* which probes *pool1*'s members' TCP service port.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol TCP --
↪protocol-port 23456 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol TCP
openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --
↪timeout 10 --type TCP pool1
```

(continues on next page)

(continued from previous page)

```
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
```

Deploy a QoS ruled load balancer

This solution limits the bandwidth available through the Load Balancer’s VIP by applying a Neutron Quality of Service(QoS) policy to the VIP, so Load Balancer can accept the QoS Policy from Neutron; Then limits the vip of Load Balancer incoming or outgoing traffic.

Note: Before using this feature, please make sure the Neutron QoS extension(qos) is enabled on running OpenStack environment by command

```
openstack extension list
```

Scenario description:

- QoS-policy created from Neutron with bandwidth-limit-rules by us.
- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an HTTP application on TCP port 80.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer and want to limit the traffic bandwidth when web traffic reaches the vip.

Solution:

1. Create QoS policy *qos-policy-bandwidth* with *bandwidth_limit* in Neutron.
2. Create load balancer *lb1* on subnet *public-subnet* with the id of *qos-policy-bandwidth*.
3. Create listener *listener1*.
4. Create pool *pool1* as *listener1*’s default pool.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
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
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet --vip-
↪qos-policy-id qos-policy-bandwidth
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 lb1 --protocol HTTP --
↪protocol-port 80
```

(continues on next page)

(continued from previous page)

```
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
openstack loadbalancer member create --subnet-id <private_subnet_id> --
↪address 192.0.2.10 --protocol-port 80 pool1
openstack loadbalancer member create --subnet-id <private_subnet_id> --
↪address 192.0.2.11 --protocol-port 80 pool1
```

Deploy a load balancer with access control list

This solution limits incoming traffic to a listener to a set of allowed source IP addresses. Any other incoming traffic will be rejected.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an custom application on TCP port 23456
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes requests to the back-end servers.
- The application on TCP port 23456 is accessible to a limited source IP addresses (192.0.2.0/24 and 198.51.100/24).

Solution:

1. Create load balancer *lb1* on subnet *public-subnet*.
2. Create listener *listener1* with allowed CIDRs.
3. Create pool *pool1* as *listener1*'s default pool.
4. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol TCP --
↪protocol-port 23456 --allowed-cidr 192.0.2.0/24 --allowed-cidr 198.51.100/
↪24 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol TCP
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
```

Deploy a non-terminated HTTPS load balancer

A non-terminated HTTPS load balancer acts effectively like a generic TCP load balancer: The load balancer will forward the raw TCP traffic from the web client to the back-end servers without decrypting it. This means that the back-end servers themselves must be configured to terminate the HTTPS connection with the web clients, and in turn, the load balancer cannot insert headers into the HTTP session indicating the client IP address. (That is, to the back-end server, all web requests will appear to originate from the load balancer.) Also, advanced load balancer features (like Layer 7 functionality) cannot be used with non-terminated HTTPS.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with a TLS-encrypted web application on TCP port 443.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes requests to the back-end servers.
- We want to employ a TCP health check to ensure that the back-end servers are available.

Solution:

1. Create load balancer *lb1* on subnet *public-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool.
4. Create a health monitor on *pool1* which probes *pool1*'s members' TCP service port.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTPS --
↪protocol-port 443 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTPS
openstack loadbalancer healthmonitor create --delay 5 --max-retries 4 --
↪timeout 10 --type HTTPS --url-path /healthcheck pool1
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
```

Deploy a TLS-terminated HTTPS load balancer

With a TLS-terminated HTTPS load balancer, web clients communicate with the load balancer over 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, we offload the CPU-intensive encryption work to the load balancer, and enable the possibility of using advanced load balancer features, like Layer 7 features and header manipulation.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with regular HTTP application on TCP port 80.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- A TLS certificate, key, and intermediate certificate chain for www.example.com have been obtained from an external certificate authority. These now exist in the files server.crt, server.key, and ca-chain.crt in the current directory. The key and certificate are PEM-encoded, and the intermediate certificate chain is multiple PEM-encoded certs concatenated together. The key is not encrypted with a passphrase.
- We want to configure a TLS-terminated HTTPS load balancer that is accessible from the internet using the key and certificate mentioned above, which distributes requests to the back-end servers over the non-encrypted HTTP protocol.
- Octavia is configured to use barbican for key management.

Solution:

1. Combine the individual cert/key/intermediates to a single PKCS12 file.
2. Create a barbican *secret* resource for the PKCS12 file. We will call this *tls_secret1*.
3. Create load balancer *lb1* on subnet *public-subnet*.
4. Create listener *listener1* as a TERMINATED_HTTPS listener referencing *tls_secret1* as its default TLS container.
5. Create pool *pool1* as *listener1*'s default pool.
6. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.
↪crt -passout pass: -out server.p12
openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e
↪'base64' --payload="$(base64 < server.p12)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --protocol-port 443 --protocol␣
↪TERMINATED_HTTPS --name listener1 --default-tls-container=$(openstack␣
↪secret list | awk '/ tls_secret1 / {print $2}') lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
```

(continues on next page)

(continued from previous page)

```
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
```

Deploy a TLS-terminated HTTPS load balancer with SNI

This example is exactly like *Deploy a TLS-terminated HTTPS load balancer*, except that we have multiple TLS certificates that we would like to use on the same listener using Server Name Indication (SNI) technology.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with regular HTTP application on TCP port 80.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- TLS certificates, keys, and intermediate certificate chains for www.example.com and www2.example.com have been obtained from an external certificate authority. These now exist in the files server.crt, server.key, ca-chain.crt, server2.crt, server2.key, and ca-chain2.crt in the current directory. The keys and certificates are PEM-encoded, and the intermediate certificate chains are multiple certs PEM-encoded and concatenated together. Neither key is encrypted with a passphrase.
- We want to configure a TLS-terminated HTTPS load balancer that is accessible from the internet using the keys and certificates mentioned above, which distributes requests to the back-end servers over the non-encrypted HTTP protocol.
- If a web client connects that is not SNI capable, we want the load balancer to respond with the certificate for www.example.com.

Solution:

1. Combine the individual cert/key/intermediates to single PKCS12 files.
2. Create barbican *secret* resources for the PKCS12 files. We will call them *tls_secret1* and *tls_secret2*.
3. Create load balancer *lb1* on subnet *public-subnet*.
4. Create listener *listener1* as a TERMINATED_HTTPS listener referencing *tls_secret1* as its default TLS container, and referencing both *tls_secret1* and *tls_secret2* using SNI.
5. Create pool *pool1* as *listener1*'s default pool.
6. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
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
```

(continues on next page)

(continued from previous page)

```
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)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
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
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
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
```

Deploy a TLS-terminated HTTPS load balancer with client authentication

With a TLS-terminated HTTPS load balancer, web clients communicate with the load balancer over 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, we offload the CPU-intensive encryption work to the load balancer, and enable the possibility of using advanced load balancer features, like Layer 7 features and header manipulation. Adding client authentication allows users to authenticate connections to the VIP using certificates. This is also known as two-way TLS authentication.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with a regular HTTP application on TCP port 80.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- A TLS certificate, key, and intermediate certificate chain for www.example.com have been obtained from an external certificate authority. These now exist in the files server.crt, server.key, and ca-chain.crt in the current directory. The key and certificate are PEM-encoded, and the intermediate certificate chain is multiple PEM-encoded certificates concatenated together. The key is not encrypted with a passphrase.
- A Certificate Authority (CA) certificate chain and optional Certificate Revocation List (CRL) have been obtained from an external certificate authority to authenticate client certificates against.
- We want to configure a TLS-terminated HTTPS load balancer that is accessible from the internet using the key and certificate mentioned above, which distributes requests to the back-end servers over the non-encrypted HTTP protocol.
- Octavia is configured to use barbican for key management.

Solution:

1. Combine the individual cert/key/intermediates to a single PKCS12 file.
2. Create a barbican *secret* resource for the PKCS12 file. We will call this *tls_secret1*.
3. Create a barbican *secret* resource for the client CA certificate. We will call this *client_ca_cert*.
4. Optionally create a barbican *secret* for the CRL file. We will call this *client_ca_crl*.
5. Create load balancer *lb1* on subnet *public-subnet*.
6. Create listener *listener1* as a TERMINATED_HTTPS listener referencing *tls_secret1* as its default TLS container, client authentication enabled, *client_ca_cert* as the client CA tls container reference, and *client_ca_crl* as the client CRL container reference.
7. Create pool *pool1* as *listener1*'s default pool.
8. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.
↪crt -passout pass: -out server.p12
openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e
↪'base64' --payload="$(base64 < server.p12)"
openstack secret store --name='client_ca_cert' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < client_ca.pem)"
openstack secret store --name='client_ca_crl' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < client_ca.crl)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --protocol-port 443 --protocol␣
↪TERMINATED_HTTPS --name listener1 --default-tls-container=$(openstack␣
↪secret list | awk '/ tls_secret1 / {print $2}') --client-
↪authentication=MANDATORY --client-ca-tls-container-ref=$(openstack secret␣
↪list | awk '/ client_ca_cert / {print $2}') --client-crl-container=
↪$(openstack secret list | awk '/ client_ca_crl / {print $2}') lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
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
```

Deploy a secure HTTP/2 load balancer with ALPN TLS extension

This example is exactly like *Deploy a TLS-terminated HTTPS load balancer*, except that we would like to enable HTTP/2 load balancing. The load balancer negotiates HTTP/2 with clients as part of the TLS handshake via the Application-Layer Protocol Negotiation (ALPN) TLS extension.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with regular HTTP application on TCP port 80.

- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- A TLS certificate, key, and intermediate certificate chain for www.example.com have been obtained from an external certificate authority. These now exist in the files server.crt, server.key, and ca-chain.crt in the current directory. The key and certificate are PEM-encoded, and the intermediate certificate chain is multiple PEM-encoded certs concatenated together. The key is not encrypted with a passphrase.
- We want to configure a TLS-terminated HTTP/2 load balancer that is accessible from the internet using the key and certificate mentioned above, which distributes requests to the back-end servers over the non-encrypted HTTP protocol.
- Octavia is configured to use barbican for key management.

Solution:

1. Combine the individual cert/key/intermediates to a single PKCS12 file.
2. Create a barbican *secret* resource for the PKCS12 file. We will call this *tls_secret1*.
3. Create load balancer *lb1* on subnet *public-subnet*.
4. Create listener *listener1* as a TERMINATED_HTTPS listener referencing *tls_secret1* as its default TLS container, and *h2* ALPN protocol ID and *http/1.1* as fall-back protocol should the client not support HTTP/2.
5. Create pool *pool1* as *listener1*'s default pool.
6. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.
↪crt -passout pass: -out server.p12
openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e
↪'base64' --payload="$(base64 < server.p12)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --protocol-port 443 --protocol␣
↪TERMINATED_HTTPS --alpn-protocol h2 --alpn-protocol http/1.1 --name␣
↪listener1 --default-tls-container=$(openstack secret list | awk '/ tls_
↪secret1 / {print $2}') lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
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
```

Deploy HTTP and TLS-terminated HTTPS load balancing on the same IP and backend

This example is exactly like *Deploy a TLS-terminated HTTPS load balancer*, except that we would like to have both an HTTP and TERMINATED_HTTPS listener that use the same back-end pool (and therefore, probably respond with the exact same content regardless of whether the web client uses the HTTP or HTTPS protocol to connect).

Please note that if you wish all HTTP requests to be redirected to HTTPS (so that requests are only served via HTTPS, and attempts to access content over HTTP just get redirected to the HTTPS listener), then please see the example in the *Layer 7 Cookbook*.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with regular HTTP application on TCP port 80.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- A TLS certificate, key, and intermediate certificate chain for www.example.com have been obtained from an external certificate authority. These now exist in the files server.crt, server.key, and ca-chain.crt in the current directory. The key and certificate are PEM-encoded, and the intermediate certificate chain is multiple PEM-encoded certs concatenated together. The key is not encrypted with a passphrase.
- We want to configure a TLS-terminated HTTPS load balancer that is accessible from the internet using the key and certificate mentioned above, which distributes requests to the back-end servers over the non-encrypted HTTP protocol.
- We also want to configure a HTTP load balancer on the same IP address as the above which serves the exact same content (ie. forwards to the same back-end pool) as the TERMINATED_HTTPS listener.

Solution:

1. Combine the individual cert/key/intermediates to a single PKCS12 file.
2. Create a barbican *secret* resource for the PKCS12 file. We will call this *tls_secret1*.
3. Create load balancer *lb1* on subnet *public-subnet*.
4. Create listener *listener1* as a TERMINATED_HTTPS listener referencing *tls_secret1* as its default TLS container.
5. Create pool *pool1* as *listener1*'s default pool.
6. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.
7. Create listener *listener2* as an HTTP listener with *pool1* as its default pool.

CLI commands:

```
openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.
↪crt -passout pass: -out server.p12
openstack secret store --name='tls_secret1' -t 'application/octet-stream' -e
↪'base64' --payload="$(base64 < server.p12)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
```

(continues on next page)

(continued from previous page)

```
openstack loadbalancer show lb1
openstack loadbalancer listener create --protocol-port 443 --protocol␣
↪TERMINATED_HTTPS --name listener1 --default-tls-container=$(openstack␣
↪secret list | awk '/ tls_secret1 / {print $2}') lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP
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
openstack loadbalancer listener create --protocol-port 80 --protocol HTTP --
↪name listener2 --default-pool pool1 lb1
```

Deploy a load balancer with backend re-encryption

This example will demonstrate how to enable TLS encryption from the load balancer to the backend member servers. Typically this is used with TLS termination enabled on the listener, but, to simplify the example, we are going to use an unencrypted HTTP listener. For information on setting up a TLS terminated listener, see the above section *Deploy a TLS-terminated HTTPS load balancer*.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an HTTPS application on TCP port 443.
- A Certificate Authority (CA) certificate chain and optional Certificate Revocation List (CRL) have been obtained from an external certificate authority to authenticate member server certificates against.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes web requests to the back-end servers.

Solution:

1. Create a barbican *secret* resource for the member CA certificate. We will call this *member_ca_cert*.
2. Optionally create a barbican *secret* for the CRL file. We will call this *member_ca_crl*.
3. Create load balancer *lb1* on subnet *public-subnet*.
4. Create listener *listener1*.
5. Create pool *pool1* as *listener1*'s default pool, that is TLS enabled, with a Certificate Authority (CA) certificate chain *member_ca_cert* to validate the member server certificate, and a Certificate Revocation List (CRL) *member_ca_crl* to check the member server certificate against.
6. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack secret store --name='member_ca_cert' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < member_ca.pem)"
openstack secret store --name='member_ca_crl' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < member_ca.crl)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP --enable-tls --ca-tls-container-ref
↪$(openstack secret list | awk '/ member_ca_cert / {print $2}') --crl-
↪container-ref $(openstack secret list | awk '/ member_ca_crl / {print $2}')
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
```

Deploy a load balancer with backend re-encryption and client authentication

This example will demostrate how to enable TLS encryption from the load balancer to the backend member servers with the load balancer being authenticated using TLS client authentication. Typically this is used with TLS termination enabled on the listener, but, to simplify the example, we are going to use an unencrypted HTTP listener. For information on setting up a TLS terminated listener, see the above section *Deploy a TLS-terminated HTTPS load balancer*.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an HTTPS application on TCP port 443.
- A Certificate Authority (CA) certificate chain and optional Certificate Revocation List (CRL) have been obtained from an external certificate authority to authenticate member server certificates against.
- A TLS certificate and key have been obtained from an external Certificate Authority (CA). The now exist in the files member.crt and member.key. The key and certificate are PEM-encoded and the key is not encrypted with a passphrase (for this example).
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes web requests to the back-end servers.

Solution:

1. Combine the member client authentication certificate and key to a single PKCS12 file.
2. Create a barbican *secret* resource for the PKCS12 file. We will call this *member_secret1*.
3. Create a barbican *secret* resource for the member CA certificate. We will call this *member_ca_cert*.
4. Optionally create a barbican *secret* for the CRL file. We will call this *member_ca_crl*.

5. Create load balancer *lb1* on subnet *public-subnet*.
6. Create listener *listener1*.
7. Create pool *pool1* as *listener1*'s default pool, that is TLS enabled, with a TLS container reference for the member client authentication key and certificate pkcs12, also with a Certificate Authority (CA) certificate chain *member_ca_cert* to validate the member server certificate, and a Certificate Revocation List (CRL) *member_ca_crl* to check the member server certificate against.
8. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openssl pkcs12 -export -inkey member.key -in member.crt -passout pass: -out␣
↪member.p12
openstack secret store --name='member_secret1' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < member.p12)"
openstack secret store --name='member_ca_cert' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < member_ca.pem)"
openstack secret store --name='member_ca_crl' -t 'application/octet-stream' -
↪e 'base64' --payload="$(base64 < member_ca.crl)"
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol HTTP --enable-tls --ca-tls-container-ref
↪$(openstack secret list | awk '/ member_ca_cert / {print $2}') --crl-
↪container-ref $(openstack secret list | awk '/ member_ca_crl / {print $2}')␣
↪--tls-container-ref $(openstack secret list | awk '/ member_secret1 /
↪{print $2}')
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
```

Deploy a UDP load balancer with a health monitor

This is a load balancer solution suitable for UDP-based services.

Scenario description:

- Back-end servers 192.0.2.10 and 192.0.2.11 on subnet *private-subnet* have been configured with an application on UDP port 1234.
- Subnet *public-subnet* is a shared external subnet created by the cloud operator which is reachable from the internet.
- We want to configure a basic load balancer that is accessible from the internet, which distributes requests to the back-end servers.
- We want to employ a UDP health check to ensure that the back-end servers are available. UDP health checks may not work correctly if ICMP Destination Unreachable (ICMP type 3) messages are blocked by a security rule (see *Other health monitors*).

Solution:

1. Create load balancer *lb1* on subnet *private-subnet*.
2. Create listener *listener1*.
3. Create pool *pool1* as *listener1*'s default pool.
4. Create a health monitor on *pool1* which connects to the back-end servers.
5. Add members 192.0.2.10 and 192.0.2.11 on *private-subnet* to *pool1*.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
# Re-run the following until lb1 shows ACTIVE and ONLINE statuses:
openstack loadbalancer show lb1
openstack loadbalancer listener create --name listener1 --protocol UDP --
↪protocol-port 1234 lb1
openstack loadbalancer pool create --name pool1 --lb-algorithm ROUND_ROBIN --
↪listener listener1 --protocol UDP
openstack loadbalancer healthmonitor create --delay 3 --max-retries 2 --
↪timeout 2 --type UDP-CONNECT pool1
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
```

Health Monitor Best Practices

An Octavia health monitor is a process that does periodic health checks on each back-end member to pre-emptively detect failed members and temporarily pull them out of the pool.

If the health monitor detects a failed member, it removes it from the pool and marks the member in ERROR. After you have corrected the member 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 members are not removed from the pool. This can lead to service disruption for web clients.

See also the command, loadbalancer healthmonitor create.

Configuration arguments for all health monitors

All health monitor types for 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 subsequent health checks a given back-end server must fail before it is considered *down*, or that a failed back-end server must pass to be considered *up* again.

Configuration arguments for HTTP health monitors

In addition to the arguments listed earlier in *Configuration arguments for all health monitors*, 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".

For a complete list of configuration arguments for Octavia health monitors, see the command, loadbalancer healthmonitor create.

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 a 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 the application is healthy and no more. This may mean ensuring database or other external storage connections are up and running, server load is acceptable, the site is not in maintenance mode, and other tests specific to your application.
- The page generated by the health check should be very light weight:
  - 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 may reference cached data. For example, you may find it useful to run a more extensive health check via cron and store the results of this 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.
- Since Octavia only cares about the HTTP status code returned, and since health checks are run so frequently, it may make sense to use the "HEAD" or "OPTIONS" HTTP methods to cut down on unnecessary processing of a whole page.

Other health monitors

Other health monitor types include `PING`, `TCP`, `HTTPS`, `SCTP`, `TLS-HELLO`, and `UDP-CONNECT`.

`PING` health monitors send periodic ICMP PING requests to the back-end servers. Obviously, your back-end servers must be configured to allow PINGs in order for these health checks to pass.

Warning: Health monitors of type `PING` only check if the member is reachable and responds to ICMP echo requests. It will not detect if your application running on that instance is healthy or not. Most pools should use one of the other health monitor options. `PING` should only be used in specific cases where an ICMP echo request is a valid health check.

`TCP` health monitors open a TCP connection to the back-end server's protocol port. Your custom TCP application should be written to respond OK to the load balancer connecting, opening a TCP connection, and closing it again after the TCP handshake without sending any data.

`HTTPS` health monitors operate exactly like HTTP health monitors, but with ssl back-end servers. Unfortunately, this causes problems if the servers are performing client certificate validation, as HAProxy won't have a valid cert. In this case, using `TLS-HELLO` type monitoring is an alternative.

`SCTP` health monitors send an INIT packet to the back-end server's port. If an application is listening on this port, the Operating System should reply with an INIT ACK packet, but if the port is closed, it replies with an ABORT packet. If the health monitor receives an INIT ACK packet, it immediatly closes the connection with an ABORT packet, and considers that the server is ONLINE.

`TLS-HELLO` health monitors simply ensure the back-end server responds to SSLv3 client hello messages. It will not check any other health metrics, like status code or body contents.

`UDP-CONNECT` health monitors do a basic UDP port connect. Health monitors of this type may not work correctly if Destination Unreachable (ICMP type 3) is not enabled on the member server or is blocked by a security rule. A member server may be marked as operating status ONLINE when it is actually down.

Intermediate certificate chains

Some TLS certificates require you to install an intermediate certificate chain in order for web client browsers to trust the certificate. This chain can take several forms, and is a file provided by the organization from whom you obtained your TLS certificate.

PEM-encoded chains

The simplest form of the intermediate chain is a PEM-encoded text file that either contains a sequence of individually-encoded PEM certificates, or a PEM encoded PKCS7 block(s). If this is the type of intermediate chain you have been provided, the file will contain either `-----BEGIN PKCS7-----` or `-----BEGIN CERTIFICATE-----` near the top of the file, and one or more blocks of 64-character lines of ASCII text (that will look like gobbedlygook to a human). These files are also typically named with a `.crt` or `.pem` extension.

DER-encoded chains

If the intermediates chain provided to you is a file that contains what appears to be random binary data, it is likely that it is a PKCS7 chain in DER format. These files also may be named with a `.p7b` extension.

You may use the binary DER file as-is when building your PKCS12 bundle:

```
openssl pkcs12 -export -inkey server.key -in server.crt -certfile ca-chain.
↪p7b -passout pass: -out server.p12
```

... or you can convert it to a series of PEM-encoded certificates:

```
openssl pkcs7 -in intermediates-chain.p7b -inform DER -print_certs -out␣
↪intermediates-chain.crt
```

... or you can convert it to a PEM-encoded PKCS7 bundle:

```
openssl pkcs7 -in intermediates-chain.p7b -inform DER -outform PEM -out␣
↪intermediates-chain.crt
```

If the file is not a PKCS7 DER bundle, either of the two `openssl pkcs7` commands will fail.

Further reading

For examples of using Layer 7 features for more advanced load balancing, please see: *Layer 7 Cookbook*

7.1.2 Layer 7 Cookbook

Introduction

This document gives several examples of common L7 load balancer usage. For a description of L7 load balancing see: *Layer 7 Load Balancing*

For the purposes of this guide we assume that the OpenStack Client command-line interface is going to be used to configure all features of Octavia with the Octavia driver back-end. Also, in order to keep these examples short, we assume that many non-L7 configuration tasks (such as deploying loadbalancers, listeners, pools, members, healthmonitors, etc.) have already been accomplished. A description of the starting conditions is given in each example below.

Examples

Redirect *http://www.example.com/* to *https://www.example.com/*

Scenario description:

- Load balancer *lb1* has been set up with `TERMINATED_HTTPS` listener *tls_listener* on TCP port 443.
- *tls_listener* has been populated with a default pool, members, etc.
- *tls_listener* is available under the DNS name *https://www.example.com/*
- We want any regular HTTP requests to TCP port 80 on *lb1* to be redirected to *tls_listener* on TCP port 443.

Solution:

1. Create listener *http_listener* as an HTTP listener on *lb1* port 80.
2. Set up an L7 Policy *policy1* on *http_listener* with action `REDIRECT_TO_URL` pointed at the URL *https://www.example.com/*
3. Add an L7 Rule to *policy1* which matches all requests.

CLI commands:

```
openstack loadbalancer listener create --name http_listener --protocol HTTP --
↪protocol-port 80 lb1
openstack loadbalancer l7policy create --action REDIRECT_PREFIX --redirect-
↪prefix https://www.example.com/ --name policy1 http_listener
openstack loadbalancer l7rule create --compare-type STARTS_WITH --type PATH --
↪value / policy1
```

(continues on next page)

(continued from previous page)

Send requests starting with /js or /images to *static_pool*

Scenario description:

- Listener *listener1* on load balancer *lb1* is set up to send all requests to its default_pool *pool1*.
- We are introducing static content servers 10.0.0.10 and 10.0.0.11 on subnet *private-subnet*, and want any HTTP requests with a URL that starts with either "/js" or "/images" to be sent to those two servers instead of *pool1*.

Solution:

1. Create pool *static_pool* on *lb1*.
2. Populate *static_pool* with the new back-end members.
3. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *static_pool*.
4. Create an L7 Rule on *policy1* which looks for "/js" at the start of the request path.
5. Create L7 Policy *policy2* with action `REDIRECT_TO_POOL` pointed at *static_pool*.
6. Create an L7 Rule on *policy2* which looks for "/images" at the start of the request path.

CLI commands:

```
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name static_pool --protocol HTTP
openstack loadbalancer member create --address 10.0.0.10 --protocol-port 80 --
↪subnet-id private-subnet static_pool
openstack loadbalancer member create --address 10.0.0.11 --protocol-port 80 --
↪subnet-id private-subnet static_pool
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool static_pool --name policy1 listener1
openstack loadbalancer l7rule create --compare-type STARTS_WITH --type PATH --
↪value /js policy1
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool static_pool --name policy2 listener1
openstack loadbalancer l7rule create --compare-type STARTS_WITH --type PATH --
↪value /images policy2
```

Alternate solution (using regular expressions):

1. Create pool *static_pool* on *lb1*.
2. Populate *static_pool* with the new back-end members.
3. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *static_pool*.
4. Create an L7 Rule on *policy1* which uses a regular expression to match either "/js" or "/images" at the start of the request path.

CLI commands:

```
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name static_pool --protocol HTTP
openstack loadbalancer member create --address 10.0.0.10 --protocol-port 80 --
↪subnet-id private-subnet static_pool
openstack loadbalancer member create --address 10.0.0.11 --protocol-port 80 --
↪subnet-id private-subnet static_pool
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool static_pool --name policy1 listener1
openstack loadbalancer l7rule create --compare-type REGEX --type PATH --value
↪'^/(js|images)' policy1
```

Send requests for *http://www2.example.com/* to *pool2*

Scenario description:

- Listener *listener1* on load balancer *lb1* is set up to send all requests to its default_pool *pool1*.
- We have set up a new pool *pool2* on *lb1* and want any requests using the HTTP/1.1 hostname *www2.example.com* to be sent to *pool2* instead.

Solution:

1. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *pool2*.
2. Create an L7 Rule on *policy1* which matches the hostname *www2.example.com*.

CLI commands:

```
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool pool2 --name policy1 listener1
openstack loadbalancer l7rule create --compare-type EQUAL_TO --type HOST_NAME␣
↪--value www2.example.com policy1
```

Send requests for **.example.com* to *pool2*

Scenario description:

- Listener *listener1* on load balancer *lb1* is set up to send all requests to its default_pool *pool1*.
- We have set up a new pool *pool2* on *lb1* and want any requests using any HTTP/1.1 hostname like **.example.com* to be sent to *pool2* instead.

Solution:

1. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *pool2*.
2. Create an L7 Rule on *policy1* which matches any hostname that ends with *example.com*.

CLI commands:

```
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool pool2 --name policy1 listener1
openstack loadbalancer l7rule create --compare-type ENDS_WITH --type HOST_
↪NAME --value example.com policy1
```

Send unauthenticated users to *login_pool* (scenario 1)

Scenario description:

- `TERMINATED_HTTPS` listener *listener1* on load balancer *lb1* is set up to send all requests to its default_pool *pool1*.
- The site behind *listener1* requires all web users to authenticate, after which a browser cookie *auth_token* will be set.
- When web users log out, or if the *auth_token* is invalid, the application servers in *pool1* clear the *auth_token*.
- We want to introduce new secure authentication server 10.0.1.10 on Neutron subnet *secure_subnet* (a different Neutron subnet from the default application servers) which handles authenticating web users and sets the *auth_token*.

Note: Obviously, to have a more secure authentication system that is less vulnerable to attacks like XSS, the new secure authentication server will need to set session variables to which the default_pool servers will have access outside the data path with the web client. There may be other security concerns as well. This example is not meant to address how these are to be accomplished--it's mainly meant to show how L7 application routing can be done based on a browser cookie.

Solution:

1. Create pool *login_pool* on *lb1*.
2. Add member 10.0.1.10 on *secure_subnet* to *login_pool*.
3. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *login_pool*.
4. Create an L7 Rule on *policy1* which looks for browser cookie *auth_token* (with any value) and matches if it is *NOT* present.

CLI commands:

```
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name login_pool --protocol HTTP
openstack loadbalancer member create --address 10.0.1.10 --protocol-port 80 --
↪subnet-id secure_subnet login_pool
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool login_pool --name policy1 listener1
openstack loadbalancer l7rule create --compare-type REGEX --key auth_token --
↪type COOKIE --value '.*' --invert policy1
```

Send unauthenticated users to *login_pool* (scenario 2)

Scenario description:

- `TERMINATED_HTTPS` listener *listener1* on load balancer *lb1* is set up to send all requests to its default_pool *pool1*.
- The site behind *listener1* requires all web users to authenticate, after which a browser cookie *auth_token* will be set.
- When web users log out, or if the *auth_token* is invalid, the application servers in *pool1* set *auth_token* to the literal string "INVALID".

- We want to introduce new secure authentication server 10.0.1.10 on Neutron subnet *secure_subnet* (a different Neutron subnet from the default application servers) which handles authenticating web users and sets the *auth_token*.

Note: Obviously, to have a more secure authentication system that is less vulnerable to attacks like XSS, the new secure authentication server will need to set session variables to which the default_pool servers will have access outside the data path with the web client. There may be other security concerns as well. This example is not meant to address how these are to be accomplished-- it's mainly meant to show how L7 application routing can be done based on a browser cookie.

Solution:

1. Create pool *login_pool* on *lb1*.
2. Add member 10.0.1.10 on *secure_subnet* to *login_pool*.
3. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *login_pool*.
4. Create an L7 Rule on *policy1* which looks for browser cookie *auth_token* (with any value) and matches if it is *NOT* present.
5. Create L7 Policy *policy2* with action `REDIRECT_TO_POOL` pointed at *login_pool*.
6. Create an L7 Rule on *policy2* which looks for browser cookie *auth_token* and matches if it is equal to the literal string "INVALID".

CLI commands:

```
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name login_pool --protocol HTTP
openstack loadbalancer member create --address 10.0.1.10 --protocol-port 80 --
↪subnet-id secure_subnet login_pool
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool login_pool --name policy1 listener1
openstack loadbalancer l7rule create --compare-type REGEX --key auth_token --
↪type COOKIE --value '.*' --invert policy1
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool login_pool --name policy2 listener1
openstack loadbalancer l7rule create --compare-type EQUAL_TO --key auth_token␣
↪--type COOKIE --value INVALID policy2
```

Send requests for *http://api.example.com/api* to *api_pool*

Scenario description:

- Listener *listener1* on load balancer *lb1* is set up to send all requests to its default_pool *pool1*.
- We have created pool *api_pool* on *lb1*, however, for legacy business logic reasons, we only want requests sent to this pool if they match the hostname *api.example.com* AND the request path starts with */api*.

Solution:

1. Create L7 Policy *policy1* with action `REDIRECT_TO_POOL` pointed at *api_pool*.
2. Create an L7 Rule on *policy1* which matches the hostname *api.example.com*.

3. Create an L7 Rule on *policy1* which matches */api* at the start of the request path. (This rule will be logically ANDed with the previous rule.)

CLI commands:

```
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool api_pool --name policy1 listener1
openstack loadbalancer l7rule create --compare-type EQUAL_TO --type HOST_NAME␣
↪--value api.example.com policy1
openstack loadbalancer l7rule create --compare-type STARTS_WITH --type PATH --
↪value /api policy1
```

Set up A/B testing on an existing production site using a cookie

Scenario description:

- Listener *listener1* on load balancer *lb1* is a production site set up as described under *Send requests starting with /js or /images to static_pool* (alternate solution) above. Specifically:
  - HTTP requests with a URL that starts with either "/js" or "/images" are sent to pool *static_pool*.
  - All other requests are sent to *listener1's* default_pool *pool1*.
- We are introducing a "B" version of the production site, complete with its own default_pool and static_pool. We will call these *pool_B* and *static_pool_B* respectively.
- The *pool_B* members should be 10.0.0.50 and 10.0.0.51, and the *static_pool_B* members should be 10.0.0.100 and 10.0.0.101 on subnet *private-subnet*.
- Web clients which should be routed to the "B" version of the site get a cookie set by the member servers in *pool1*. This cookie is called "site_version" and should have the value "B".

Solution:

1. Create pool *pool_B* on *lb1*.
2. Populate *pool_B* with its new back-end members.
3. Create pool *static_pool_B* on *lb1*.
4. Populate *static_pool_B* with its new back-end members.
5. Create L7 Policy *policy2* with action `REDIRECT_TO_POOL` pointed at *static_pool_B*. This should be inserted at position 1.
6. Create an L7 Rule on *policy2* which uses a regular expression to match either "/js" or "/images" at the start of the request path.
7. Create an L7 Rule on *policy2* which matches the cookie "site_version" to the exact string "B".
8. Create L7 Policy *policy3* with action `REDIRECT_TO_POOL` pointed at *pool_B*. This should be inserted at position 2.
9. Create an L7 Rule on *policy3* which matches the cookie "site_version" to the exact string "B".

A word about L7 Policy position: Since L7 Policies are evaluated in order according to their position parameter, and since the first L7 Policy whose L7 Rules all evaluate to True is the one whose action is followed, it is important that L7 Policies with the most specific rules get evaluated first.

For example, in this solution, if *policy3* were to appear in the listener's L7 Policy list before *policy2* (that is, if *policy3* were to have a lower position number than *policy2*), then if a web client were to request the URL http://www.example.com/images/a.jpg with the cookie "site_version:B", then *policy3* would match, and the load balancer would send the request to *pool_B*. From the scenario description, this request clearly was meant to be sent to *static_pool_B*, which is why *policy2* needs to be evaluated before *policy3*.

CLI commands:

```
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name pool_B --protocol HTTP
openstack loadbalancer member create --address 10.0.0.50 --protocol-port 80 --
↪subnet-id private-subnet pool_B
openstack loadbalancer member create --address 10.0.0.51 --protocol-port 80 --
↪subnet-id private-subnet pool_B
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name static_pool_B --protocol HTTP
openstack loadbalancer member create --address 10.0.0.100 --protocol-port 80 -
↪-subnet-id private-subnet static_pool_B
openstack loadbalancer member create --address 10.0.0.101 --protocol-port 80 -
↪-subnet-id private-subnet static_pool_B
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool static_pool_B --name policy2 --position 1 listener1
openstack loadbalancer l7rule create --compare-type REGEX --type PATH --value
↪'^/(js|images)' policy2
openstack loadbalancer l7rule create --compare-type EQUAL_TO --key site_
↪version --type COOKIE --value B policy2
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool pool_B --name policy3 --position 2 listener1
openstack loadbalancer l7rule create --compare-type EQUAL_TO --key site_
↪version --type COOKIE --value B policy3
```

Redirect requests with an invalid TLS client authentication certificate

Scenario description:

- Listener *listener1* on load balancer *lb1* is configured for `OPTIONAL` client_authentication.
- Web clients that do not present a TLS client authentication certificate should be redirected to a signup page at *http://www.example.com/signup*.

Solution:

1. Create the load balancer *lb1*.
2. Create a listener *listner1* of type `TERMINATED_TLS` with a client_ca_tls_container_ref and client_authentication `OPTIONAL`.
3. Create a L7 Policy *policy1* on *listener1* with action `REDIRECT_TO_URL` pointed at the URL *http://www.example.com/signup*.
4. Add an L7 Rule to *policy1* that does not match `SSL_CONN_HAS_CERT`.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
openstack loadbalancer listener create --name listener1 --protocol TERMINATED_
↪HTTPS --client-authentication OPTIONAL --protocol-port 443 --default-tls-
↪container-ref http://192.0.2.15:9311/v1/secrets/697c2a6d-ffbe-40b8-be5e-
↪7629fd636bca --client-ca-tls-container-ref http://192.0.2.15:9311/v1/
↪secrets/dba60b77-8dad-4171-8a96-f21e1ca5fb46 lb1
openstack loadbalancer l7policy create --action REDIRECT_TO_URL --redirect-
↪url http://www.example.com/signup --name policy1 listener1
openstack loadbalancer l7rule create --type SSL_CONN_HAS_CERT --invert --
↪compare-type EQUAL_TO --value True policy1
```

Send users from the finance department to pool2

Scenario description:

- Users from the finance department have client certificates with the OU field of the distinguished name set to `finance`.
- Only users with valid finance department client certificates should be able to access `pool2`. Others will be rejected.

Solution:

1. Create the load balancer *lb1*.
2. Create a listener *listner1* of type `TERMINATED_TLS` with a client_ca_tls_container_ref and client_authentication `MANDATORY`.
3. Create a pool *pool2* on load balancer *lb1*.
4. Create a L7 Policy *policy1* on *listener1* with action `REDIRECT_TO_POOL` pointed at *pool2*.
5. Add an L7 Rule to *policy1* that matches `SSL_CONN_HAS_CERT`.
6. Add an L7 Rule to *policy1* that matches `SSL_VERIFY_RESULT` with a value of 0.
7. Add an L7 Rule to *policy1* of type `SSL_DN_FIELD` that looks for "finance" in the "OU" field of the client authentication distinguished name.

CLI commands:

```
openstack loadbalancer create --name lb1 --vip-subnet-id public-subnet
openstack loadbalancer listener create --name listener1 --protocol TERMINATED_
↪HTTPS --client-authentication MANDATORY --protocol-port 443 --default-tls-
↪container-ref http://192.0.2.15:9311/v1/secrets/697c2a6d-ffbe-40b8-be5e-
↪7629fd636bca --client-ca-tls-container-ref http://192.0.2.15:9311/v1/
↪secrets/dba60b77-8dad-4171-8a96-f21e1ca5fb46 lb1
openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --loadbalancer␣
↪lb1 --name pool2 --protocol HTTP
openstack loadbalancer l7policy create --action REDIRECT_TO_POOL --redirect-
↪pool pool2 --name policy1 listener1
openstack loadbalancer l7rule create --type SSL_CONN_HAS_CERT --compare-type␣
↪EQUAL_TO --value True policy1
```

(continues on next page)

(continued from previous page)

```
openstack loadbalancer l7rule create --type SSL_VERIFY_RESULT --compare-type␣
↪EQUAL_TO --value 0 policy1
openstack loadbalancer l7rule create --type SSL_DN_FIELD --compare-type EQUAL_
↪TO --key OU --value finance policy1
```

7.2 Guides

7.2.1 Layer 7 Load Balancing

What is L7 load balancing?

Layer 7 load balancing takes its name from the OSI model, indicating that the load balancer distributes requests to back-end pools based on layer 7 (application) data. Layer 7 load balancing is also known as "request switching," "application load balancing," "content based routing," "content based switching," and "content based balancing."

A layer 7 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/optimized to serve specific types of content. For example, one group of back-end servers (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, layer 7 load balancing does not require that all pools behind the load balancing service have the same content. In fact, it is generally expected that a layer 7 load balancer expects the back-end servers from different pools will have different content. Layer 7 load balancers are capable of directing requests based on URI, host, HTTP headers, and other data in the application message.

L7 load balancing in Octavia

The layer 7 load balancing capabilities described in this document were added to Neutron LBaaS and Octavia in the Mitaka release cycle (Octavia 0.8).

While layer 7 load balancing in general can theoretically be done for any well-defined layer 7 application interface, for the purposes of Octavia, L7 functionality refers only to the HTTP protocol and its semantics.

How does it work?

Neutron LBaaS and Octavia accomplish the logic of layer 7 load balancing through the use of L7 Rules and L7 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.

These concepts and their specific details are expanded upon below.

L7 Rules

An L7 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.

See also: Octavia API Reference

Rule types

L7 rules have the following types:

- `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. (eg. "txt", "jpg", etc.)
- `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 the certificate is valid.
- `SSL_VERIFY_RESULT`: This rule will match the TLS client authentication certificate validation result. A value of '0' means the certificate was successfully validated. A value greater than '0' means the certificate failed validation. This value follows the openssl-verify result codes.
- `SSL_DN_FIELD`: The rule looks for a Distinguished Name feild defined in the key parameter and compares it against the value parameter in the rule.

Comparison types

L7 rules of a given type always do comparisons. The types of comparisons we support 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

Invert

In order to more fully express the logic required by some policies, rules may have their result inverted. That is to say, 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", and an inverted "regex" rule returns true only if the given regex does not match.)

L7 Policies

An L7 Policy is a collection of L7 rules associated with a Listener, and which may also have an association to a back-end pool. Policies describe actions that should be taken by the load balancing software if all of the rules in the policy return true.

See also: Octavia API Reference

Policy Logic

Policy logic is very simple: All the rules associated with a given policy are logically ANDed together. A request must match all the policy's 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, possibly, by making a more elaborate regular expression).

Policy Actions

If an L7 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_TO_POOL`: The request is forwarded to the back-end pool associated with the L7 policy.

Policy Position

When multiple L7 Policies are associated with a listener, then the policies' `position` parameter becomes important. The `position` parameter is used when determining the order in which L7 policies are evaluated. Here are a few notes about how policy position affects listener behavior:

- In the reference implementation (haproxy amphorae) of Octavia, 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 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 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.

L7 usage examples

For a cookbook of common L7 usage examples, please see the *Layer 7 Cookbook*

Useful links

- Octavia API Reference
- LBaaS Layer 7 rules
- Using ACLs and fetching samples
- OpenSSL openssl-verify command

7.2.2 Octavia Provider Feature Matrix

Load Balancer Features

Provider feature support matrix for an Octavia load balancer.

Load Balancer API Features

These features are documented in the Octavia API reference Create a Load Balancer section. Summary

Feature	*Status*	**Amphora Provider**	**OVN Provider**
admin_state_up	mandatory	✓	✓
availability_zone	optional	✓	×
description	optional	✓	✓
flavor	optional	✓	×
name	optional	✓	✓
Load Balancer statistics	mandatory	✓	×
Load Balancer status tree	mandatory	✓	✓
tags	optional	✓	✓
vip_address	optional	✓	✓
vip_network_id	optional	✓	✓
vip_port_id	optional	✓	✓
vip_qos_policy_id	optional	✓	✓
vip_subnet_id	optional	✓	✓

Details

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer create [--enable | --disable] <load_balancer>`

  Notes: Enables and disables the load balancer.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **availability_zone Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--availability-zone <availability_zone>] <load_balancer>`

  Notes: The availability zone to deploy the load balancer into.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **description Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--description <description>] <load_balancer>`

  Notes: The description of the load balancer. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **flavor Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--flavor <flavor>] <load_balancer>`

  Notes: The flavor of the load balancer.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **name Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--name <name>] <load_balancer>`

  Notes: The name of the load balancer. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **Load Balancer statistics Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer stats show <load_balancer>`

  Notes: The ability to show statistics for a load balancer.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **Load Balancer status tree Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer status show <load_balancer>`

  Notes: The ability to show a status tree for the load balancer.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--tag <tag>] <load_balancer>`

  Notes: The tags for the load balancer. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`

  - **OVN Provider:** `complete`

- **vip_address Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--vip-address <vip_address>] <load_balancer>`

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **vip_network_id Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--vip-network-id <vip_network_id>] <load_balancer>`

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **vip_port_id Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--vip-port-id <vip_port_id>] <load_balancer>`

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **vip_qos_policy_id Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--vip-qos-policy-id <vip_qos_policy_id>] <load_balancer>`

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **vip_subnet_id Status: optional.**

  CLI commands:

  - `openstack loadbalancer create [--vip-subnet-id <vip_subnet_id>] <load_balancer>`

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

Notes:

- **This document is a continuous work in progress**

Listener Features

Provider feature support matrix for an Octavia load balancer listener.

Listener API Features

These features are documented in the Octavia API reference Create a Listener section. Summary

Feature	*Status*	**Amphora Provider**	**OVN Provider**
admin_state_up	mandatory	✓	✓
allowed_cidr	optional	✓	×
alpn_protocol	optional	✓	×
client_authentication	optional	✓	×
client_ca_tls_container_ref	optional	✓	×
client_crl_container_ref	optional	✓	×
connection_limit	optional	✓	×
default_tls_container_ref	optional	✓	×
description	optional	✓	✓
insert_headers - X-Forwarded-For	optional	✓	×
insert_headers - X-Forwarded-Port	optional	✓	×
insert_headers - X-Forwarded-Proto	optional	✓	×
insert_headers - X-SSL-Client-Verify	optional	✓	×
insert_headers - X-SSL-Client-Has-Cert	optional	✓	×
insert_headers - X-SSL-Client-DN	optional	✓	×
insert_headers - X-SSL-Client-CN	optional	✓	×
insert_headers - X-SSL-Issuer	optional	✓	×
insert_headers - X-SSL-Client-SHA1	optional	✓	×
insert_headers - X-SSL-Client-Not-Before	optional	✓	×
insert_headers - X-SSL-Client-Not-After	optional	✓	×
name	optional	✓	✓
protocol - HTTP	optional	✓	×
protocol - HTTPS	optional	✓	×
protocol - TCP	optional	✓	✓
protocol - TERMINATED_HTTPS	optional	✓	×
protocol - UDP	optional	✓	✓
protocol - SCTP	optional	✓	×
protocol_port	mandatory	✓	✓
sni_container_refs	optional	✓	×
Listener statistics	mandatory	✓	×
tags	optional	✓	✓
timeout_client_data	optional	✓	×
timeout_member_connect	optional	✓	×
timeout-member-data	optional	✓	×
timeout-tcp-inspect	optional	✓	×

continues on next page

Table 1 – continued from previous page

Feature	*Status*	Amphora Provider	OVN Provider
tls_ciphers	optional	✓	×
tls_versions	optional	✓	×

Details

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer listener create [--enable | --disable] <loadbalancer>`

  Notes: Enables and disables the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **allowed_cidr Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--allowed-cidr <allowed_cidr>] <loadbalancer>`

  Notes: CIDR to allow access to the listener (can be set multiple times).

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **alpn_protocol Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--alpn-protocol <protocol>] <loadbalancer>`

  Notes: List of accepted ALPN protocols (can be set multiple times).

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **client_authentication Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--client-authentication {NONE,OPTIONAL,MANDATORY}] <listener>`

  Notes: The TLS client authentication mode.

  Driver Support:

  - **Amphora Provider:** `complete`

  - **OVN Provider:** `missing`

- **client_ca_tls_container_ref Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--client-ca-tls-container-ref <container_ref>] <listener>`

  Notes: The ref of the key manager service secret containing a PEM format client CA certificate bundle for TERMINATED_TLS listeners.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **client_crl_container_ref Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--client-crl-container-ref <client_crl_container_ref>] <listener>`

  Notes: The URI of the key manager service secret containing a PEM format CA revocation list file for TERMINATED_TLS listeners.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **connection_limit Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--connection-limit <limit>] <listener>`

  Notes: The maximum number of connections permitted for this listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **default_tls_container_ref Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--default-tls-container-ref <container_ref>] <listener>`

  Notes: The URI of the key manager service secret containing a PKCS12 format certificate/key bundle for TERMINATED_TLS listeners.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **description Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--description <description>] <loadbalancer>`

  Notes: The description of the listener. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **insert_headers - X-Forwarded-For Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-Forwarded-For=true] <loadbalancer>`

  Notes: When true a X-Forwarded-For header is inserted into the request to the backend member that specifies the client IP address.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-Forwarded-Port Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-Forwarded-Port=true] <loadbalancer>`

  Notes: When true a X-Forwarded-Port header is inserted into the request to the backend member that specifies the listener port.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-Forwarded-Proto Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-Forwarded-Proto=true] <loadbalancer>`

  Notes: When true a X-Forwarded-Proto header is inserted into the request to the backend member.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-Verify Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-Verify=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-Verify header is inserted into the request to the backend member that contains 0 if the client authentication was successful, or an result error number greater than 0 that align to the openssl veryify error codes.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-Has-Cert Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-Has-Cert=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-Has-Cert header is inserted into the request to the backend member that is true if a client authentication certificate was presented, and false if not. Does not indicate validity.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-DN Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-DN=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-DN header is inserted into the request to the backend member that contains the full Distinguished Name of the certificate presented by the client.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-CN Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-CN=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-CN header is inserted into the request to the backend member that contains the Common Name from the full Distinguished Name of the certificate presented by the client.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Issuer Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Issuer=true] <loadbalancer>`

  Notes: When true a X-SSL-Issuer header is inserted into the request to the backend member that contains the full Distinguished Name of the client certificate issuer.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-SHA1 Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-SHA1=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-SHA1 header is inserted into the request to the backend member that contains the SHA-1 fingerprint of the certificate presented by the client in hex string format.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-Not-Before Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-Not-Before=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-Not-Before header is inserted into the request to the backend member that contains the start date presented by the client as a formatted string YYMMDDhh-mmss[Z].

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **insert_headers - X-SSL-Client-Not-After Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--insert-headers X-SSL-Client-Not-Aftr=true] <loadbalancer>`

  Notes: When true a X-SSL-Client-Not-After header is inserted into the request to the backend member that contains the end date presented by the client as a formatted string YYMMDDhh-mmss[Z].

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **name Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--name <name>] <loadbalancer>`

  Notes: The name of the load balancer listener. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol - HTTP Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol HTTP <loadbalancer>`

  Notes: HTTP protocol support for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - HTTPS Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol HTTPS <loadbalancer>`

  Notes: HTTPS protocol support for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - TCP Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol TCP <loadbalancer>`

  Notes: TCP protocol support for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol - TERMINATED_HTTPS Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol TERMINATED_HTTPS <loadbalancer>`

  Notes: Terminated HTTPS protocol support for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - UDP Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol UDP <loadbalancer>`

  Notes: UDP protocol support for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol - SCTP Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol SCTP <loadbalancer>`

  Notes: SCTP protocol support for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol_port Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer listener create --protocol-port <port> <loadbalancer>`

  Notes: The protocol port number for the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **sni_container_refs Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--sni-container-refs [<container_ref> [<container_ref> ...]]] <loadbalancer>`

  Notes: A list of URIs to the key manager service secrets containing PKCS12 format certificate/key bundles for TERMINATED_TLS listeners.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **Listener statistics Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer listener stats show <listener>`

  Notes: The ability to show statistics for a listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--tags <tag>] <loadbalancer>`

  Notes: The tags for the load balancer listener. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **timeout_client_data Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--timeout-client-data <timeout>] <loadbalancer>`

  Notes: Frontend client inactivity timeout in milliseconds.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **timeout_member_connect Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--timeout-member-connect <timeout>] <loadbalancer>`

  Notes: Backend member connection timeout in milliseconds.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **timeout-member-data Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--timeout-member-data <timeout>] <loadbalancer>`

  Notes: Backend member inactivity timeout in milliseconds.

  Driver Support:

  - **Amphora Provider:** `complete`

  - **OVN Provider:** `missing`

- **timeout-tcp-inspect Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--timeout-tcp-inspect <timeout>] <loadbalancer>`

  Notes: Time, in milliseconds, to wait for additional TCP packets for content inspection.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tls_ciphers Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--tls-ciphers <ciphers>] <loadbalancer>`

  Notes: List of accepted TLS ciphers.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tls_versions Status: optional.**

  CLI commands:

  - `openstack loadbalancer listener create [--tls-versions <versions>] <loadbalancer>`

  Notes: List of accepted TLS protocol versions.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

Notes:

- **This document is a continuous work in progress**

Pool Features

Provider feature support matrix for an Octavia load balancer pool.

Pool API Features

These features are documented in the Octavia API reference Create a Pool section. Summary

Feature	*Status*	**Amphora Provider**	**OVN Provider**
admin_state_up	mandatory	✓	✓
alpn_protocol	optional	✓	×
ca_tls_container_ref	optional	✓	×
crl_container_ref	optional	✓	×
lb_algorithm - LEAST_CONNECTIONS	optional	✓	×
lb_algorithm - ROUND_ROBIN	optional	✓	×
lb_algorithm - SOURCE_IP	optional	✓	×
lb_algorithm - SOURCE_IP_PORT	optional	×	✓
description	optional	✓	✓
name	optional	✓	✓
protocol - HTTP	optional	✓	×
protocol - HTTPS	optional	✓	×
protocol - PROXY	optional	✓	×
protocol - PROXYV2	optional	✓	×
protocol - TCP	optional	✓	✓
protocol - UDP	optional	✓	✓
protocol - SCTP	optional	✓	×
session_persistence - APP_COOKIE	optional	✓	×
session_persistence - cookie_name	optional	✓	×
session_persistence - HTTP_COOKIE	optional	✓	×
session_persistence - persistence_timeout	optional	✓	×
session_persistence - persistence_granularity	optional	✓	×
session_persistence - SOURCE_IP	optional	✓	×
tags	optional	✓	✓
tls_ciphers	optional	✓	×
tls_container_ref	optional	✓	×
tls_enabled	optional	✓	×
tls_versions	optional	✓	×

Details

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer pool create [--enable | --disable] --listener <listener>`

  Notes: Enables and disables the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **alpn_protocol Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--alpn-protocol <protocol>] --listener <listener>`

  Notes: List of accepted ALPN protocols (can be set multiple times).

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **ca_tls_container_ref Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--ca-tls-container-ref <ca_tls_container_ref>] --listener <listener>`

  Notes: The reference of the key manager service secret containing a PEM format CA certificate bundle for tls_enabled pools.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **crl_container_ref Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--crl-container-ref <crl_container_ref>] --listener <listener>`

  Notes: The reference of the key manager service secret containing a PEM format CA revocation list file for tls_enabled pools.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **lb_algorithm - LEAST_CONNECTIONS Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --lb-algorithm LEAST_CONNECTIONS --listener <listener>`

  Notes: The pool will direct connections to the member server with the least connections in use.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **lb_algorithm - ROUND_ROBIN Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --lb-algorithm ROUND_ROBIN --listener <listener>`

  Notes: The pool will direct connections to the next member server, one after the other, rotating through the available memeber servers.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **lb_algorithm - SOURCE_IP Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --lb-algorithm SOURCE_IP --listener <listener>`

  Notes: The pool will direct connections to the member server based on a hash of the source IP.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **lb_algorithm - SOURCE_IP_PORT Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --lb-algorithm SOURCE_IP_PORT --listener <listener>`

  Notes: The pool will direct connections to the member server based on a hash of the source IP and Port.

  Driver Support:

  - **Amphora Provider:** `missing`
  - **OVN Provider:** `complete`

- **description Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--description <description>] --listener <listener>`

  Notes: The description of the pool. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **name Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--name <name>] --listener <listener>`

Notes: The name of the pool. Provided by the Octavia API service.

Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol - HTTP Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol HTTP --listener <listener>`

  Notes: HTTP protocol support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - HTTPS Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol HTTP --listener <listener>`

  Notes: HTTPS protocol support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - PROXY Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol PROXY --listener <listener>`

  Notes: PROXY protocol support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - PROXYV2 Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol PROXYV2 --listener <listener>`

  Notes: PROXY protocol version 2 support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **protocol - TCP Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol TCP --listener <listener>`

  Notes: TCP protocol support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol - UDP Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol UDP --listener <listener>`

  Notes: UDP protocol support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol - SCTP Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --protocol SCTP --listener <listener>`

  Notes: SCTP protocol support for the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **session_persistence - APP_COOKIE Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --session-persistence type=APP_COOKIE --listener <listener>`

  Notes: Session persistence using an application supplied cookie.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **session_persistence - cookie_name Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --session-persistence cookie_name=chocolate --listener <listener>`

Notes: The name of the application cookie to use for session persistence.

Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **session_persistence - HTTP_COOKIE Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --session-persistence type=HTTP_COOKIE --listener <listener>`

  Notes: Session persistence using a cookie created by the load balancer.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **session_persistence - persistence_timeout Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --session-persistence persistence_timeout=360 --listener <listener>`

  Notes: The timeout, in seconds, after which a SCTP or UDP flow may be rescheduled to a different member.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **session_persistence - persistence_granularity Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --session-persistence persistence_granularity=255.255.255.255 --listener <listener>`

  Notes: The netmask used to determine SCTP or UDP SOURCE_IP session persistence.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **session_persistence - SOURCE_IP Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create --session-persistence type=SOURCE_IP --listener <listener>`

  Notes: Session persistence using the source IP address.

  Driver Support:

  - **Amphora Provider:** `complete`

  - **OVN Provider:** `missing`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--tag <tag>] --listener <listener>`

  Notes: The tags for the pool. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **tls_ciphers Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--tls-ciphers <ciphers>] --listener <listener>`

  Notes: List of TLS ciphers available for member connections.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tls_container_ref Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--tls-container-ref <container-ref>] --listener <listener>`

  Notes: The reference to the key manager service secret containing a PKCS12 format certificate/key bundle for tls_enabled pools for TLS client authentication to the member servers.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tls_enabled Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--enable-tls] --listener <listener>`

  Notes: When true connections to backend member servers will use TLS encryption.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tls_versions Status: optional.**

  CLI commands:

  - `openstack loadbalancer pool create [--tls-versions <versions>] --listener <listener>`

  Notes: List of TLS protocol versions available for member connections.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

Notes:

- **This document is a continuous work in progress**

Member Features

Provider feature support matrix for an Octavia load balancer member.

Member API Features

These features are documented in the Octavia API reference Create a Member section. Summary

Feature	*Status*	**Amphora Provider**	**OVN Provider**
admin_state_up	mandatory	✓	✓
address	mandatory	✓	✓
backup	optional	✓	×
Batch update members	mandatory	✓	✓
monitor_address	optional	✓	×
monitor_port	optional	✓	×
name	optional	✓	✓
protocol_port	mandatory	✓	✓
subnet_id	optional	✓	✓
tags	optional	✓	✓
weight	optional	✓	×

Details

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer member create [--enable | --disable] <pool>`

  Notes: Enables and disables the member.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **address Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer member create --address <ip_address> <pool>`

Notes: The IP address for the member.

Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **backup Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--enable-backup] <pool>`

  Notes: True if the member is a backup member server.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **Batch update members Status: mandatory.**

  Notes: Ability to update the members of a pool in one API call.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `partial` **Notes:** The OVN provider does not support all of the member features.

- **monitor_address Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--monitor-address <monitor_address>] <pool>`

  Notes: An alternate IP address used for health monitoring a backend member.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **monitor_port Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--monitor-port <monitor_port>] <pool>`

  Notes: An alternate protocol port used for health monitoring a backend member.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **name Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--name <name>] <pool>`

  Notes: The name for the member. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **protocol_port Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer member create --protocol_port <protocol_port> <pool>`

  Notes: The protocol port number to connect with on the member server.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **subnet_id Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--subnet-id <subnet_id>] <pool>`

  Notes: The subnet ID the member service is accessible from.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--tag <tag>] <pool>`

  Notes: The tags for the member. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `complete`

- **weight Status: optional.**

  CLI commands:

  - `openstack loadbalancer member create [--weight <weight>] <pool>`

  Notes: The weight of a member determines the portion of requests or connections it services compared to the other members of the pool.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

Notes:

- **This document is a continuous work in progress**

Health Monitor Features

Provider feature support matrix for an Octavia load balancer health monitor.

Health Monitor API Features

These features are documented in the Octavia API reference Create a Health Monitor section. Summary

Feature	*Status*	Amphora Provider	OVN Provider
admin_state_up	mandatory	✓	×
delay	mandatory	✓	×
domain_name	optional	✓	×
expected_codes	optional	✓	×
http_method	optional	✓	×
http_version	optional	✓	×
name	optional	✓	×
max_retries	mandatory	✓	×
max_retries_down	optional	✓	×
tags	optional	✓	×
timeout	mandatory	✓	×
type - HTTP	optional	✓	×
type - HTTPS	optional	✓	×
type - PING	optional	✓	×
type - TCP	optional	✓	×
type - TLS-HELLO	optional	✓	×
type - UDP-CONNECT	optional	✓	×
type - SCTP	optional	✓	×
url_path	optional	✓	×

Details

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--enable | --disable] <pool>`

  Notes: Enables and disables the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **delay Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --delay <delay> <pool>`

  Notes: The time, in seconds, between sending probes to members.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **domain_name Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--domain-name <domain_name>] <pool>`

  Notes: The domain name, which be injected into the HTTP Host Header to the backend server for HTTP health check.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **expected_codes Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--expected-codes <codes>] <pool>`

  Notes: The list of HTTP status codes expected in response from the member to declare it healthy.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **http_method Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--http-method <HTTP_METHOD>] <pool>`

  Notes: The HTTP method that the health monitor uses for requests.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **http_version Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [[--http-version <http_version>]] <pool>`

  Notes: The HTTP version to use for health checks.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **name Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--name <name>] <pool>`

  Notes: The name of the health monitor. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **max_retries Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --max-retries <max_retries> <pool>`

  Notes: The number of successful checks before changing the operating status of the member to ONLINE.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **max_retries_down Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--max-retries-down <max_retries_down>] <pool>`

  Notes: The number of allowed check failures before changing the operating status of the member to ERROR.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--tag <tag>] <pool>`

  Notes: The tags for the health monitor. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **timeout Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --timeout <timeout> <pool>`

  Notes: The maximum time, in seconds, that a monitor waits to connect before it times out.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - HTTP Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type HTTP <pool>`

  Notes: Use HTTP for the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - HTTPS Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type HTTPS <pool>`

  Notes: Use HTTPS for the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - PING Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type PING <pool>`

  Notes: Use PING for the health monitor.

  Driver Support:

  - **Amphora Provider:** `partial` **Notes:** CentOS 7 based amphora do not support PING health monitors.
  - **OVN Provider:** `missing`

- **type - TCP Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type TCP <pool>`

  Notes: Use TCP for the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - TLS-HELLO Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type TLS-HELLO <pool>`

  Notes: Use TLS-HELLO handshake for the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - UDP-CONNECT Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type UDP-CONNECT <pool>`

  Notes: Use UDP-CONNECT for the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - SCTP Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create --type SCTP <pool>`

  Notes: Use SCTP for the health monitor.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **url_path Status: optional.**

  CLI commands:

  - `openstack loadbalancer healthmonitor create [--url-path <url_path>] <pool>`

  Notes: The HTTP URL path of the request sent by the monitor to test the health of a backend member.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

Notes:

- **This document is a continuous work in progress**

L7 Policy Features

Provider feature support matrix for an Octavia load balancer L7 Policies.

L7 Policy API Features

These features are documented in the Octavia API reference Create an L7 Policy section. Summary

Feature	*Status*	**Amphora Provider**	**OVN Provider**
action - REDIRECT_TO_POOL	optional	✓	×
action - REDIRECT_TO_PREFIX	optional	✓	×
action - REDIRECT_TO_URL	optional	✓	×
action - REJECT	optional	✓	×
admin_state_up	mandatory	✓	×
description	optional	✓	×
name	optional	✓	×
position	optional	✓	×
redirect_http_code	optional	✓	×
redirect_pool_id	optional	✓	×
redirect_prefix	optional	✓	×
redirect_url	optional	✓	×
tags	optional	✓	×

Details

- **action - REDIRECT_TO_POOL Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create --action REDIRECT_TO_POOL <listener>`

  Notes: The L7 policy action REDIRECT_TO_POOL.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **action - REDIRECT_TO_PREFIX Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create --action REDIRECT_TO_PREFIX <listener>`

  Notes: The L7 policy action REDIRECT_TO_PREFIX.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **action - REDIRECT_TO_URL Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create --action REDIRECT_TO_URL <listener>`

  Notes: The L7 policy action REDIRECT_TO_URL.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **action - REJECT Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create --action REJECT <listener>`

  Notes: The L7 policy action REJECT.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--enable | --disable] <listener>`

  Notes: Enables and disables the L7 policy.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **description Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--description <description>] <listener>`

  Notes: The description of the L7 policy. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **name Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--name <name>] <listener>`

  Notes: The name of the L7 policy. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`

  - **OVN Provider:** `missing`

- **position Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--position <position>] <listener>`

  Notes: The position of this policy on the listener.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **redirect_http_code Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--redirect-http-code <redirect_http_code>] <listener>`

  Notes: Requests matching this policy will be redirected to the specified URL or Prefix URL with the HTTP response code.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **redirect_pool_id Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--redirect-pool <pool>] <listener>`

  Notes: Requests matching this policy will be redirected to the pool with this ID.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **redirect_prefix Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--redirect-prefix <url>] <listener>`

  Notes: Requests matching this policy will be redirected to this Prefix URL.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **redirect_url Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--redirect-url <url>] <listener>`

  Notes: Requests matching this policy will be redirected to this URL.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7policy create [--tag <tag>] <listener>`

  Notes: The tags for the L7 policy. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

Notes:

- **This document is a continuous work in progress**

L7 Rule Features

Provider feature support matrix for an Octavia load balancer L7 Rules.

L7 Rule API Features

These features are documented in the Octavia API reference Create an L7 Rule section. Summary

Feature	*Status*	Amphora Provider	OVN Provider
admin_state_up	mandatory	✓	×
compare_type - CONTAINS	mandatory	✓	×
compare_type - ENDS_WITH	mandatory	✓	×
compare_type - EQUAL_TO	mandatory	✓	×
compare_type - REGEX	mandatory	✓	×
compare_type - STARTS_WITH	mandatory	✓	×
invert	optional	✓	×
key	optional	✓	×
tags	optional	✓	×
type - COOKIE	optional	✓	×
type - FILE_TYPE	optional	✓	×
type - HEADER	optional	✓	×
type - HOST_NAME	optional	✓	×
type - PATH	optional	✓	×
type - SSL_CONN_HAS_CERT	optional	✓	×
type - SSL_VERIFY_RESULT	optional	✓	×
type - SSL_DN_FIELD	optional	✓	×
value	mandatory	✓	×

Details

- **admin_state_up Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create [--enable | --disable] <l7policy>`

  Notes: Enables and disables the L7 rule.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **compare_type - CONTAINS Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create --compare-type CONTAINS <l7policy>`

  Notes: The CONTAINS comparison type for the L7 rule.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **compare_type - ENDS_WITH Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create --compare-type ENDS_WITH <l7policy>`

  Notes: The ENDS_WITH comparison type for the L7 rule.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **compare_type - EQUAL_TO Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create --compare-type EQUAL_TO <l7policy>`

  Notes: The EQUAL_TO comparison type for the L7 rule.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **compare_type - REGEX Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create --compare-type REGEX <l7policy>`

Notes: The REGEX comparison type for the L7 rule.

Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **compare_type - STARTS_WITH Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create --compare-type STARTS_WITH <l7policy>`

  Notes: The STARTS_WITH comparison type for the L7 rule.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **invert Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create [--invert] <l7policy>`

  Notes: When true the logic of the rule is inverted.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **key Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create [--key <key>] <l7policy>`

  Notes: The key to use for the comparison.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **tags Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create [--tag <tag>] <l7policy>`

  Notes: The tags for the L7 rule. Provided by the Octavia API service.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - COOKIE Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type COOKIE <l7policy>`

  Notes: The COOKIE L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - FILE_TYPE Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type FILE_TYPE <l7policy>`

  Notes: The FILE_TYPE L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - HEADER Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type HEADER <l7policy>`

  Notes: The HEADER L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - HOST_NAME Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type HOST_NAME <l7policy>`

  Notes: The HOST_NAME L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - PATH Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type PATH <l7policy>`

  Notes: The PATH L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - SSL_CONN_HAS_CERT Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type SSL_CONN_HAS_CERT <l7policy>`

  Notes: The SSL_CONN_HAS_CERT L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - SSL_VERIFY_RESULT Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type SSL_VERIFY_RESULT <l7policy>`

  Notes: The SSL_VERIFY_RESULT L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **type - SSL_DN_FIELD Status: optional.**

  CLI commands:

  - `openstack loadbalancer l7rule create --type SSL_DN_FIELD <l7policy>`

  Notes: The SSL_DN_FIELD L7 rule type.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

- **value Status: mandatory.**

  CLI commands:

  - `openstack loadbalancer l7rule create --value <value> <l7policy>`

  Notes: The value to use for the comparison.

  Driver Support:

  - **Amphora Provider:** `complete`
  - **OVN Provider:** `missing`

Notes:

- **This document is a continuous work in progress**

7.3 References

7.3.1 Octavia Software Development Kits (SDK)

Introduction

This is a list of known SDKs and language bindings that support OpenStack load balancing via the Octavia API. This list is a "best effort" to keep updated, so please check with your favorite SDK project to see if they support OpenStack load balancing. If not, open a bug for them!

Note: The projects listed here may not be maintained by the OpenStack LBaaS team. Please submit bugs for these projects through their respective bug tracking systems.

Go

Gophercloud

Java

OpenStack4j

Python

OpenStack SDK

7.4 Videos