This presentation provides an overview and details related to the Neutron networking service in OpenStack.
Neutron is a core component of OpenStack that virtualizes network components, emulating the physical network. Networks can be created on request from the CLI, REST API, or Dashboard UI.

Neutron provides support to build networks/subnetworks and routers plus advanced network topologies and policies for load balancing, firewalls,or VPN.

Neutron provides the networking objects, including vendor support through the use of plugins (pluggable python classes), that the other OpenStack components use.
There are many network vendors. That leads to many Neutron plugins that can be downloaded from the OpenStack Marketplace,git,or directly from the vendor.

Objectives

At the end of this presentation, you should be able to:

- Understand Neutron concepts and plugin architecture
  - Modular Layer (ML2) *type* and *mechanism* drivers
  - Agents: IPAM, L3, DHCP
- Understand Open vSwitch network implementation
- Explain what network namespaces are and why they are important
- Describe Octavia LBaaS v2
- Use the CLI to implement a load balancer solution
- Explain what a floating IP is and how to allocate/associate
- Describe NAT and packet filtering, including how Neutron security groups apply

Network management with Neutron

What is Neutron's job?

Neutron overview (1)

- Using Neutron, you can create **virtual networks**:
  - Networks (public/external and private/internal/tenant)
  - Subnetworks
    - VM instances deployed to subnets
  - Routers
- Provides function for:
  - Layer 2 switching and layer 3 routing
  - IP address management (IPAM)
  - DHCP
  - IP address translation (SNAT / DNAT) and packet filtering
  - Security groups

MIRANTIS

Neutron consists of the **neutron-server**, a database for persistent storage, and any number of plugin agents, which provide other services such as interfacing with native Linux networking mechanisms, external devices, or SDN controllers.

Neutron provides control plane functions (layer-2 and layer-3) - layer 2 connectivity between workloads of the same tenant plus L2 isolation between workloads of different tenants.
IPAM and the use of Linux namespaces provide support for overlapping IPs, which is critical if you are providing cloud services.
L3/NAT forwarding allows external network access to deployed VMs using floating IP addresses.
The DHCP service uses **dnsmasq** by default.
Neutron security groups behave similar to a virtual firewall at the port level.

Neutron overview (2)

- In addition, you can add other networking extensions, such as:
  - Load Balancer (OpenStack Octavia)
  - Firewall
  - VPN
- Neutron is *vendor agnostic* using plugins (ML2, L3, DHCP, LB, FW, VPN)
- Neutron provides *reference implementations* of each plugin
  - For example, dnsmasq is the DHCP reference implementation

MIRANTIS

Neutron uses a plugin model to manifest a tenant's logical network layout to *any physical network infrastructure*. Vendor plugins can be downloaded from the OpenStack Marketplace, github, or a vendor web site. Using vendor plugins allows mapping of the same logical network layout to a different (vendor specific) physical network infrastructure.

OpenStack provides default plugins, commonly referred to as *reference implementations*.

Neutron abstractions and architecture

Terminology

- Network
  - An isolated L2 segment, analogous to VLAN in the physical networking world
- Subnet
  - One or more per network
  - Used to allocate IP addresses when new ports are created on a network; can be automatically allocated from a predefined pool
  - Can be entire subnet or a subset of IP address range
  - VM instances are deployed to a specified subnet (default behavior)
- Port
  - Connection point for attaching a single device, such as the NIC of a VM, to a virtual network
  - Optionally, you can specify the port when deploying a VM instance
- Router
  - Provides virtual layer-3 services such as routing and NAT between self-service and provider networks or among self-service networks belonging to a project
  - Uses a layer-3 agent to manage routers in network namespaces (qrouter-)
- Security group
  - Provides definitions for virtual firewall rules that control ingress (inbound to instances) and egress (outbound from instances) network traffic at the port level

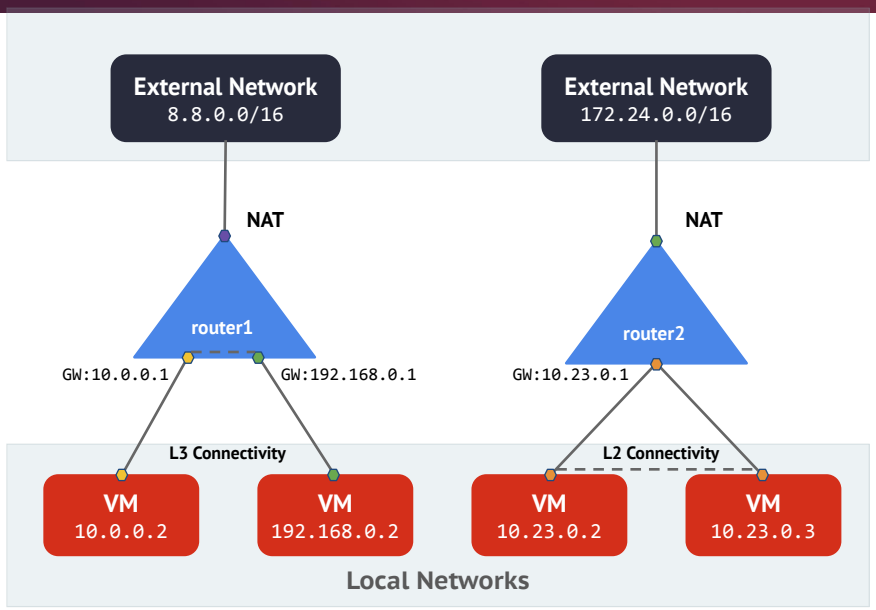
This slide contains standard network terminology.
Neutron security groups might be new to you, but their function is not. Security groups provide packet filtering rules. Neutron provides a default security group that blocks all ingress traffic to all VM instances. For example, if you require SSH access to VM instances, you need to create a security group rule that allows ingress traffic to port 22.

Neutron API features

FEATURE	NOTES
Local network	Create/read/delete/update networks
External or Shared networks	User can *read*, must have **admin** role to create/delete/update
VM connectivity	Attach VMs to any readable ports
Subnet CRUD	Create multiple subnets on a network, manage IPs, gateway
Port CRUD	Create port on a network, with IP from any subnet on it
Router CRUD	Add/remove networks, routes

- Users (*member* role) can perform almost all actions
- Must have **admin** role to manage external (public) or shared networks

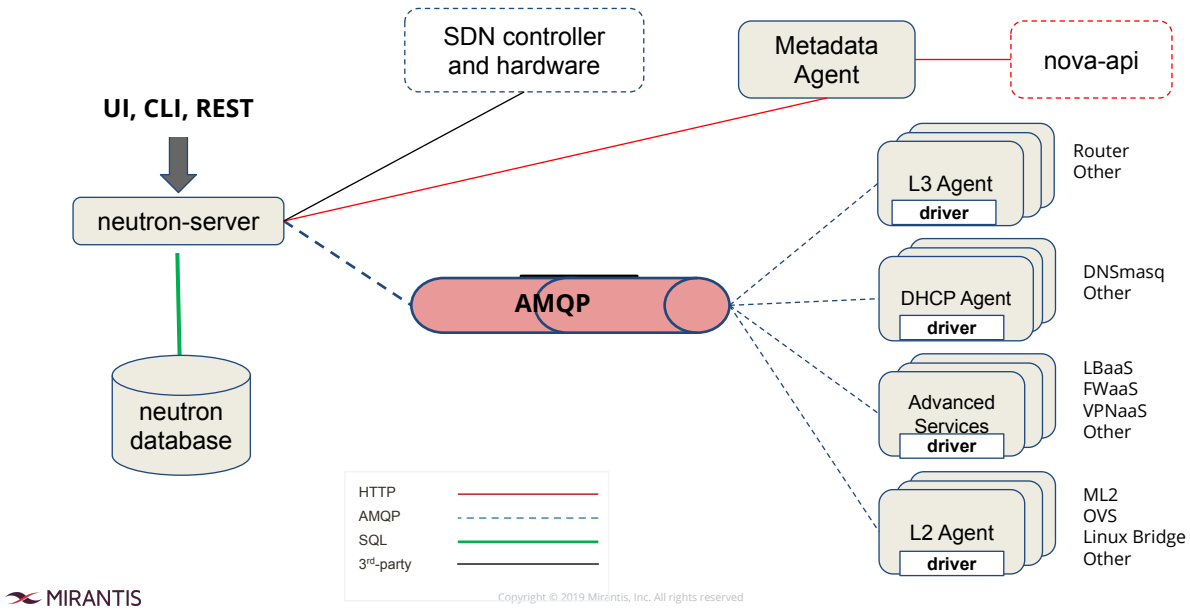
Example network with Neutron



From the user standpoint, Neutron offers several network abstractions which aim to deliver the same look and feel as real resources in a datacenter, but for cloud instances.

Each neutron user is free to do the following within their quota:
- create a number of layer 2 nets
- assign them arbitrary IP pools
- create own routers
- plug routers to external (datacenter) & internal cloud networks
- Neutron routers automatically NAT between local addresses and external networks

Neutron architecture



This slide provides an introduction to the Neutron component architecture:

- neutron-server: main API point, handles both core and extension (API and plugins) functions. It also enforces the network model and IP addressing of each port. The neutron-server and plugin agents require access to a database for persistent storage and access to a message queue for inter-communication.
  - Extension plugins are used to implement routers (L3 function), LBaaS, FWaaS, VPNaaS.
  - Quotas and security groups are also implemented as service extensions.
- Neutron Open vSwitch (OVS) agent: (example of an L2 agent) Manages OVS and OVS processes
- Neutron DHCP agent: Manages dnsmasq processes. Provides DHCP services to tenant networks. This agent is the same across all plugins and is responsible for maintaining DHCP configuration. The neutron-dhcp-agent requires message queue access
- Neutron L3 agent: Provides L3/NAT forwarding for external network access of VMs on tenant networks. Requires message queue access. *Optional depending on plugin.*
- Metadata server: (not shown) Interact with metadata from nova on provisioning to assign port and IP address for VM provisioning
- Service plug-ins: Provide or manage extra capabilities such as load balancer, firewall, or VPN support
- **network provider services** (SDN server/services): Provide additional networking services that are provided to tenant networks. These SDN services may interact with the neutron-server, neutron-plugin, and/or plugin-agents via REST APIs or other communication channels.
- ML2 plugin provides network type drivers as well as mechanism drivers. Mechanism drivers are used to implement the types of networks. The supported network types are:
  - Local: Single network; can only be realized on a single host. This is only used in proof-of-concept or development environments.
  - Flat: Single network, no segregation – all servers are able to see the same broadcast traffic and can contact each other without requiring a router.
  - VLAN: Network segregation with VLAN and VLAN tagging; communication at L2 level (limitation of 4K VLAN IDs)
  - VXLAN (Virtual Extensible LAN): Same basic function as VLAN, but supports extended address ranges beyond VLAN limit
  - GRE (Generic Routing Encapsulation): Overlay network with tunneling protocol for transporting L3 data over IP

In a multiple node implementation, assuming an Open vSwitch implementation:

- Neutron-server with the L2 agents run on the controller node
- OVS agent runs on the compute node, along with nova-compute
- Metadata, L3 (routing), DHCP, and OVS agents run on the network node

neutron-server notes

neutron-server has multiple roles

- API services: handles all API requests, such as
  - Create network
  - Allocate IP to instance
- Core (layer 2) services provided by ML2 plugin
- Extensions (layers 3-7, other services) provided by specific plugins:
  - LBaaS
  - FWaaS
  - VPNaaS
  - Etc.

Neutron
Modular Layer 2 Plugin

ML2

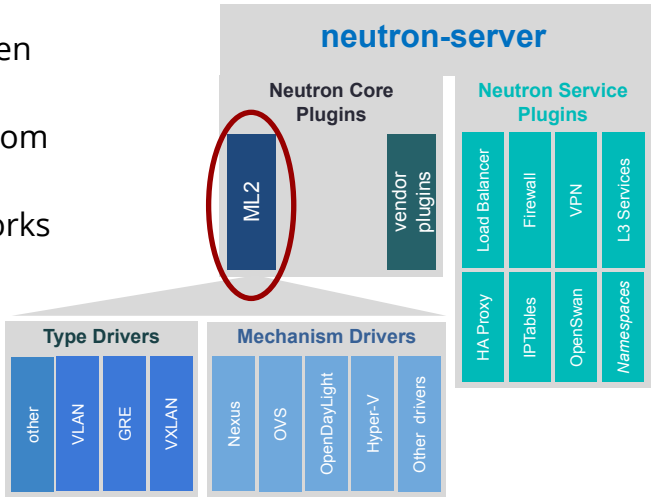
Customers have many different networking requirements. Some are simple, others are more advanced and require a level of sophistication.

- Multi-tiered Web applications
- Isolation between their customer networks for security (for example, for Cloud Service Providers)
- Might require a specific IP address scheme,
- And so on

The ML2 plugin allows Neutron to be *vendor/technology agnostic*. Neutron exposes the virtual network to the plugin and the plugin implements the network through *mechanism drivers*.

Neutron architecture: Modular layer 2 (ML2) plugin

- Provides layer 2 networking
- *Reference implementation* uses Open vSwitch (OVS)
- Separates network device types from *vendor-specific* implementations (mechanisms) to access the networks
- Network types and mechanisms become *pluggable drivers*
- Supported types:
  - VLAN, VXLAN, GRE, local, flat



The Modular layer-2 (ML2) plugin is a *framework* allowing OpenStack networking to simultaneously utilize the variety of layer-2 networking technologies found in complex real-world data centers. The ML2 framework is intended to greatly simplify adding support for new L2 networking technologies, requiring much less initial and ongoing effort than would be required to add a new monolithic core plugin. It currently works with the existing mechanism drivers for Open vSwitch, Linux bridge, SRIOV (single root input/output virtualization), MacVTap, L2 population, and hyper-v agents.

Type Drivers:

The **type managers and drivers** provide the support for network types, such as VLAN or VXLAN. Each available network type is implemented and managed by an ML2 *type driver*. Type drivers maintain any needed type-specific network state, and perform provider network validation and tenant network allocation.

The ML2 plugin currently includes drivers for the local, flat, vlan, gre and vxlan network types.

Mechanism Drivers:

The **mechanism manager and drivers** provide support for the implementation of vendor specific functions, such as the Cisco Nexus switch.

Multiple mechanism drivers can be used simultaneously to access different ports of the same virtual network. Mechanism drivers can utilize L2 agents via RPC and/or use mechanism drivers to interact with external devices or controllers. The mechanism driver is responsible for taking the information established by the type driver and ensuring that it is properly applied given the specific networking mechanisms that have been enabled. The mechanism driver interface currently supports the creation, update, and deletion of network and port resources.

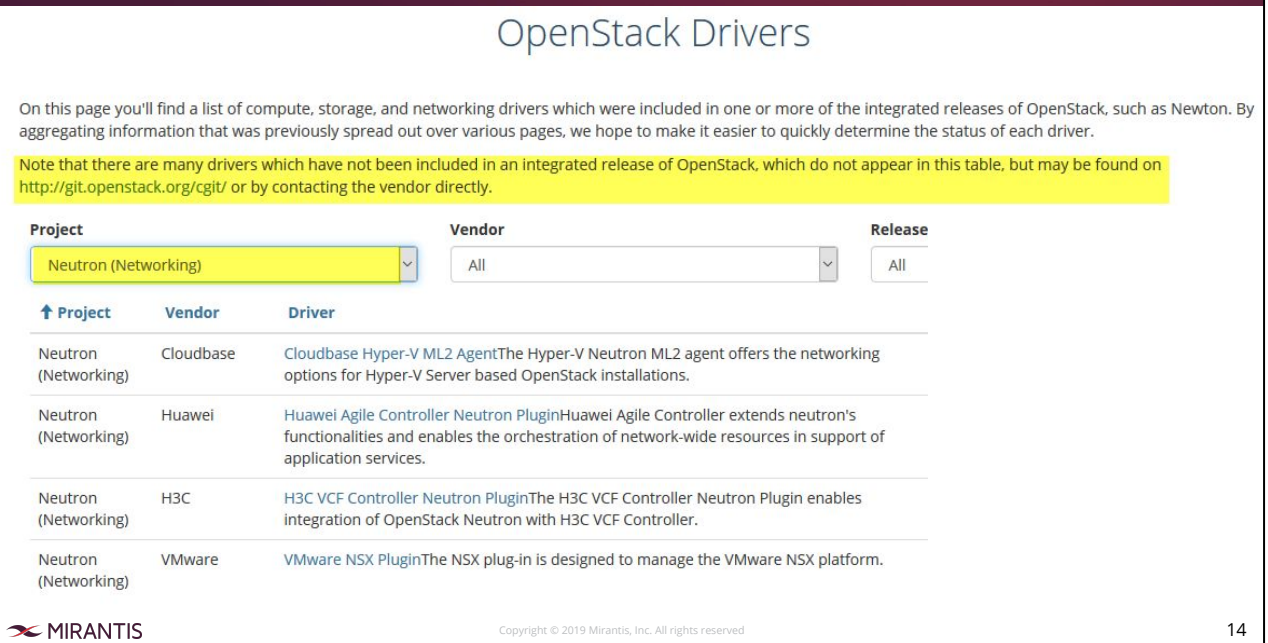
Mechanism drivers fall into one of three categories: agent based, controller based, or switch based.

The type and mechanism drivers are defined in the **ml2_conf.ini** file, for example:

type_drivers = flat,vlan,vxlan,gre

mechanism_drivers = ovs,l2pop,cisco_nexus,cisco_apic,…

ML2 plugin vendor drivers in the Marketplace



OpenStack Drivers

On this page you'll find a list of compute, storage, and networking drivers which were included in one or more of the integrated releases of OpenStack, such as Newton. By aggregating information that was previously spread out over various pages, we hope to make it easier to quickly determine the status of each driver.

Note that there are many drivers which have not been included in an integrated release of OpenStack, which do not appear in this table, but may be found on http://git.openstack.org/cgit/ or by contacting the vendor directly.

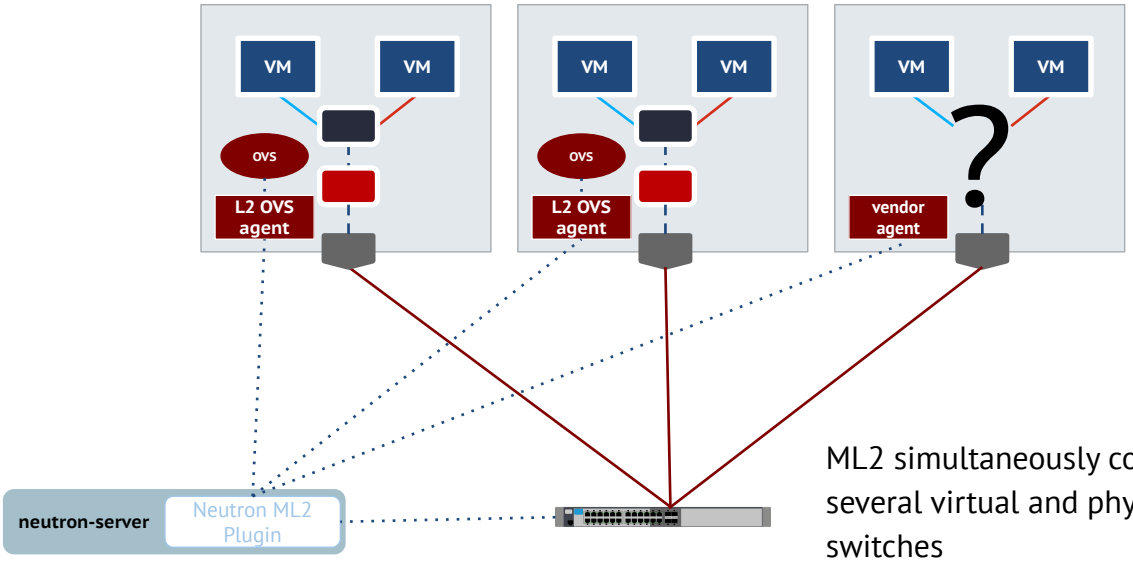
Project: Neutron (Networking) | Vendor: All | Release: All

Project	Vendor	Driver
Neutron (Networking)	Cloudbase	Cloudbase Hyper-V ML2 AgentThe Hyper-V Neutron ML2 agent offers the networking options for Hyper-V Server based OpenStack installations.
Neutron (Networking)	Huawei	Huawei Agile Controller Neutron PluginHuawei Agile Controller extends neutron's functionalities and enables the orchestration of network-wide resources in support of application services.
Neutron (Networking)	H3C	H3C VCF Controller Neutron PluginThe H3C VCF Controller Neutron Plugin enables integration of OpenStack Neutron with H3C VCF Controller.
Neutron (Networking)	VMware	VMware NSX PluginThe NSX plug-in is designed to manage the VMware NSX platform.

Neutron provides a *reference implementation* of the ML2 plugin,using Open vSwitch (OVS).
The ML2 plugin allows Neutron to be vendor agnostic with regards to how the networks are implemented. Several plugins are shown on this slide, taken from: https://www.openstack.org/marketplace/drivers/#project=neutron%20(networking)

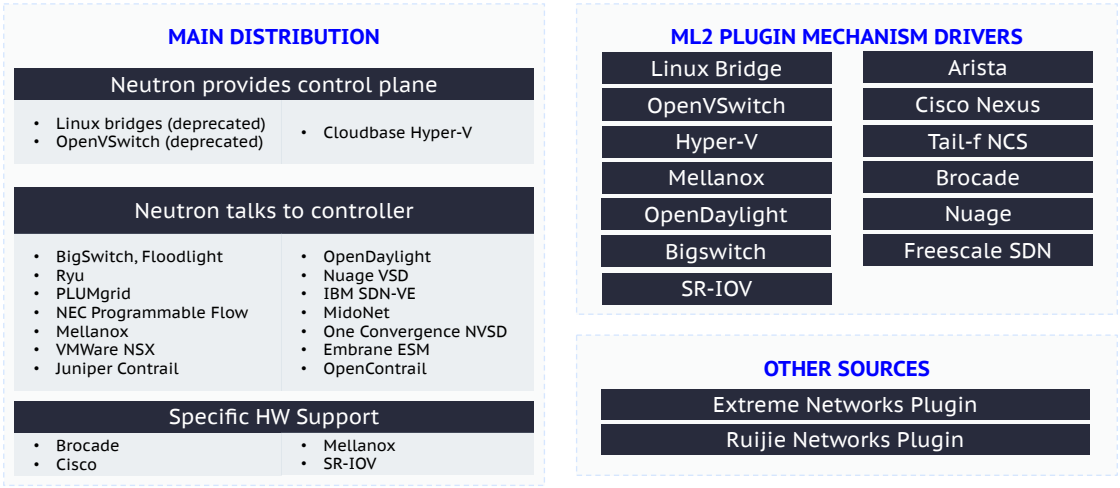
Plugins are defined in **neutron.conf**. You can run multiple plugins.
core_plugin = ml2

ML2 deployment



ML2 simultaneously controls several virtual and physical switches

Available Neutron plugins



Different plugins provide different functionality. That is the purpose of supporting vendor plugins.
Layer 2 networking is provided by the ML2 plugin: network type (flat, VLAN, VXLAN, GRE) and mechanism drivers

The most common implementation is known as the Classic Open vSwitch *reference implementation*, using the Neutron open-vswitch agent for the OVS support.

Network type and mechanism drivers:
https://github.com/openstack/neutron/tree/stable/rocky/neutron/plugins/ml2/drivers

List of vendors, drivers, and contacts:
https://wiki.openstack.org/wiki/Neutron_Plugins_and_Drivers

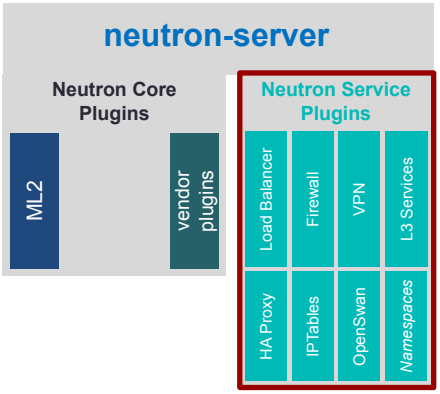
Note: Some of the plugins provide not just L2, but also L3, LB, and FW support.

Neutron
network services extensions

Neutron API Extension Model

Neutron architecture: service extensions

- Neutron service plugins provide additional services for layers 3-7
  - Routers
  - Load balancer
  - Firewall
  - VPN
  - etc

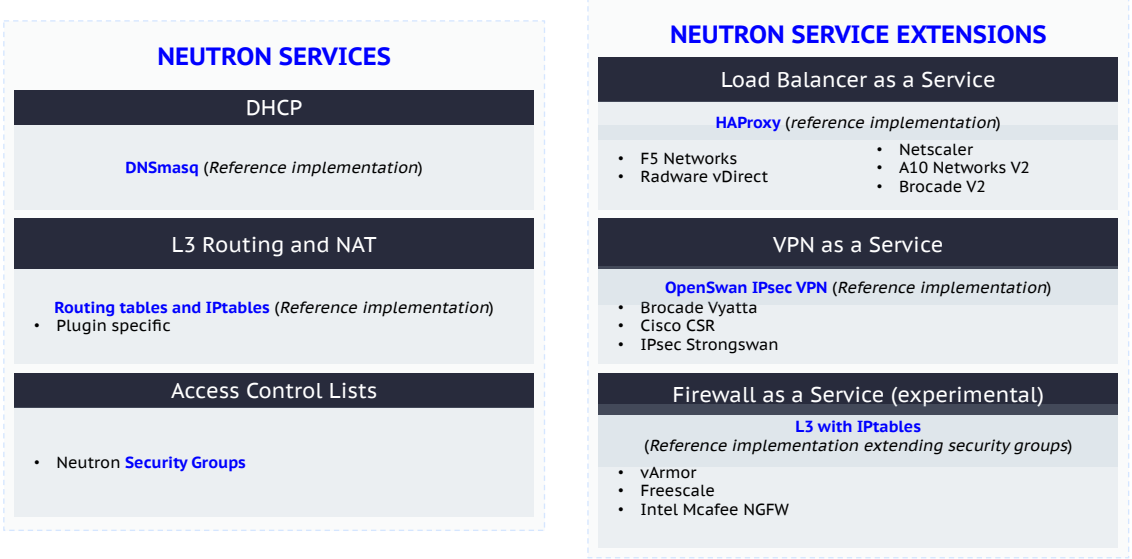


While not related to the ML2 plugin, it is important to understand the service plugins for layers 3-7.

Service plugins are defined in **neutron.conf**. You can run multiple plugins. For example:

service_plugins = router, lbaasv2

Network services drivers



Neutron provides virtual networks for your cloud. There are several networking options available, called *reference implementations*. The Neutron architecture uses pluggable agents. The most common agents are L3 (layer-3), DHCP (dynamic host IP addressing), and a layer-2 (ML2) plug-in agent.

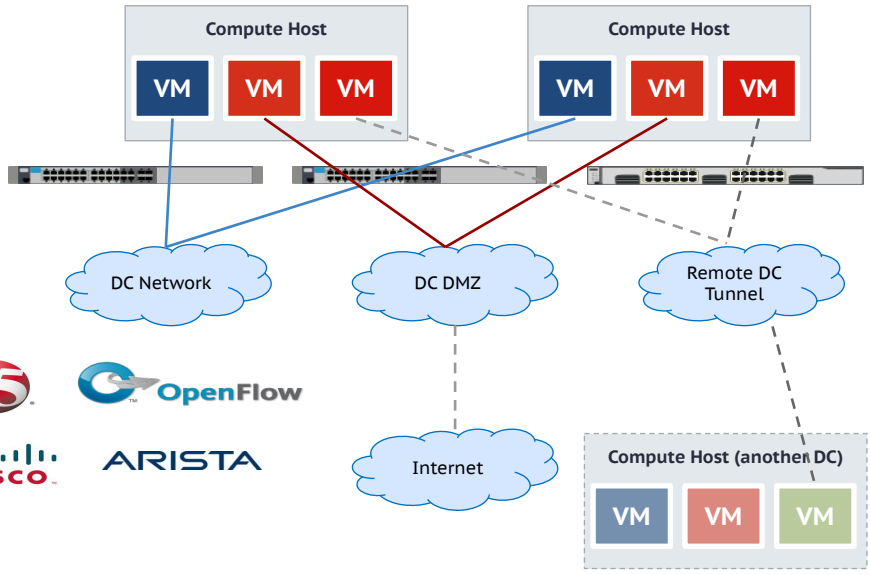
OpenStack networking also enables customers to create advanced virtual network topologies which may include services such as a firewall (FWaaS), a load balancer (LBaaS) implemented by the Octavia project, and a virtual private network (VPNaaS). Neutron provides a LBaaS *reference implementation*, using HAproxy, that can be replaced by vendor plugins, such as the F5 Networks (f5lbaas) driver.

One of the primary points of Neutron is that it's software defined networking; a lot of things that used to be done on dedicated hardware, you can now virtualize using software, and if you can do it using software, you can offer it as a service.

For more information:
https://docs.openstack.org/neutron/rocky/admin/index.html
https://docs.openstack.org/neutron/rocky/configuration/index.html
https://docs.openstack.org/neutron/rocky/admin/#

Neutron hardware diversity support

virtual networks delivered on top of datacenter hardware



nicira

OPEN VSWITCH
An Open Virtual Switch

OpenFlow

paloalto NETWORKS

cisco

ARISTA

These virtual resources are realized on top of diverse hardware and different enterprise networks (local, DMZ, remote datacenters)

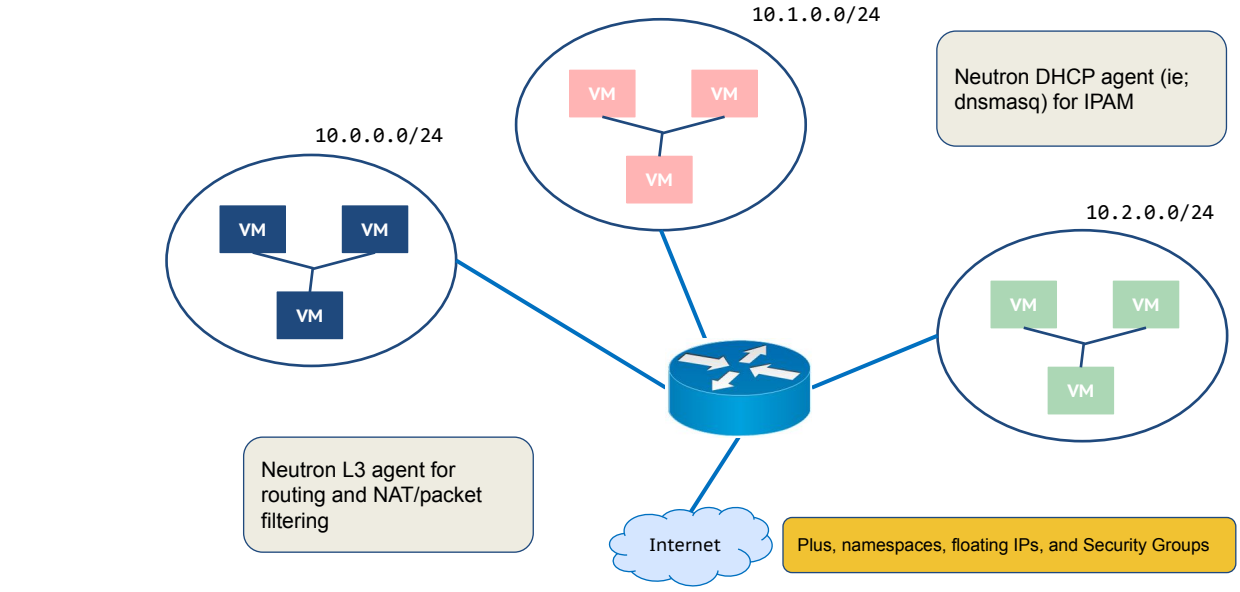
Neutron Agents

IPAM

L3 Routing

DHCP

Example: 3 subnets with 1 router



Within each individual subnet, layer 2 connectivity is available. IP addresses are assigned (IPAM) using Neutron DHCP agent, which uses **dnsmasq** by default. You can specify other IPAM tools. Each DHCP agent correlates to a *dhcp namespace*.
To connect the subnets, as well as the public/external network (the Internet), you need a router. The Neutron L3 agent provides the routing functions. In this example, the router has 4 interfaces. Each router correlates to a Linux *router namespace*. Namespaces are discussed soon.

In addition, you might need to connect to 1, or more, of the VM instances from the external/public network. This requires:

- Floating IP address (an address on the external/public network)
- Neutron Security Group (firewall) rules to control port-level access

IPAM and DHCP

- IP address management (IPAM) provides a pluggable service for allocating and managing
  - Fixed IP addresses (assigned to ports of an instance)
  - Floating IP addresses
  - Address ranges (subnets)
- Plugins enable the integration of alternate IPAM implementations or third-party IPAM systems
- Integrates with DHCP and DNS services
  - Neutron DHCP agent uses **dnsmasq**

The IPAM subsystem in Neutron allows flexible control over the lifecycle of network resources, such as:

- Fixed IP addresses assigned to Neutron ports, including DNS updates
- Floating IP addresses
- Network address ranges, defined at the sub-network level

IPAM integrates DHCP and DNS, which allows for changes in one to be seen by the other.

The IP address management driver is defined in **neutron.conf**:
Neutron IPAM (IP address management) driver to use. By default, the reference
implementation of the Neutron IPAM driver is used. (string value)
#ipam_driver = internal

L3 agent

- Neutron L3-agent:
  - Supports creation and management of **routers**
  - Connecting multiple networks
    - Public (external)
    - Private (internal)
  - Each router has a unique ***qrouter-*** namespace associated with it
    - Functions as virtual router
    - Creates NAT-ed connections to external (public) network
    - *qg-* interface: connects to public (external) network
    - *qr-* interface: connects to a private (internal) network

Network namespaces

What are namespaces?

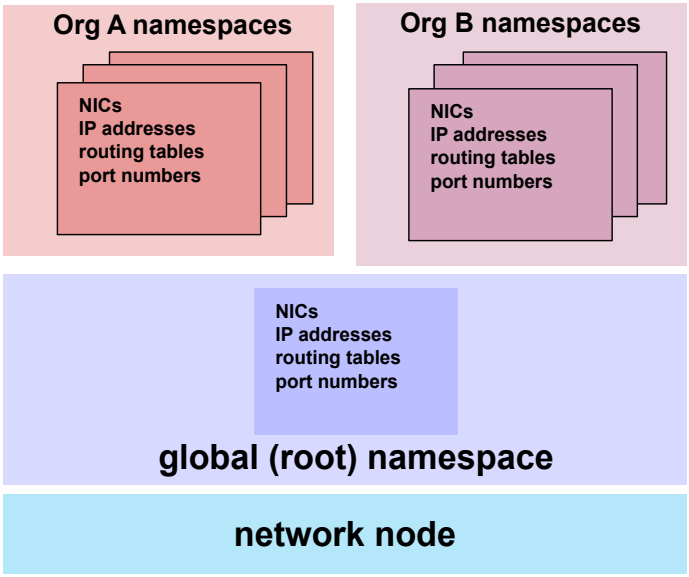
Why are they important with Neutron?

Namespaces are a Linux concept.
OpenStack is designed with multi-tenancy in mind; providing users with the ability to create and manage their own compute and network resources. Neutron supports each tenant having multiple private networks, routers, firewalls, load balancers, and other network resources.

A **network namespace** is defined as a logical copy of the network stack with its own routes, firewall rules, and network interface devices. Neutron is able to provide isolated DHCP and routing services to each tenant network through the use of namespaces.

Network namespaces

- Address challenges caused by *multi-tenancy*
  - Supports overlapping IP addresses
  - Provides network isolation (segregation) between different tenant networks
- Each router has its own namespace
- Each subnet with DHCP has its own namespace
- Automatically created when network resources (subnetwork, router, load balancer) are created
  - Has its own NICs, IP addresses, IP routing table, port numbers, and so on
  - Separate from the *global (root) namespace*, which owns global routing table



Network namespaces provide a segmentation of networking resources into different containers. The containers are created in a specialized iproute package that is installed in the network node.
The namespaces are created dynamically by Neutron when you create the network resources: network, subnetwork, port, router.
Namespaces provide multiple layer-2 networks, isolated for each tenant (project), with the capability to use overlapping IP addresses. Each namespace holds different sets of network resources. Multiple layer-2 networks, separated by routing tables - allows multiple instances of routing table to coexist on network node
Project flows are separated by VLAN ID (subnet). Internally, this is done through OpenFlow rules.

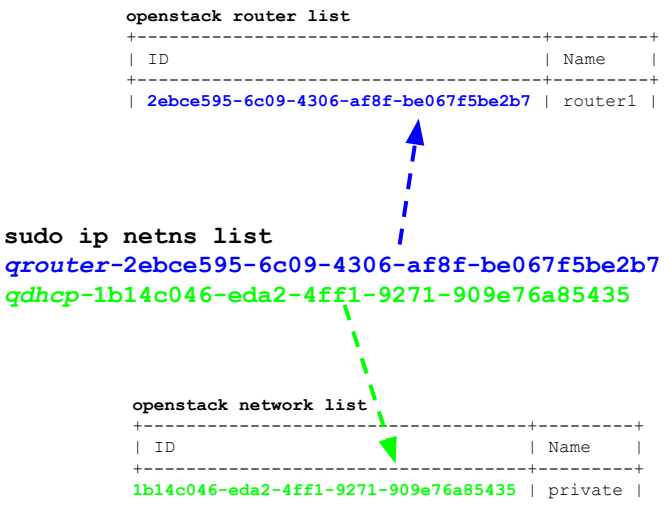
Neutron uses several namespaces. Core support creates a **qdhcp** namespace for each subnetwork and a **qrouter** namespace for each virtual router.
Extensions create additional namespaces. For example, a **qdhcp** namespace is created for Octavia Load Balancer resources.

The global namespace (sometimes called the *root namespace*) owns the physical resources.

The **ip netns** command is used to manage the namespaces.

Core namespaces

- Router namespace:
  - Every neutron router has its own qrouter namespace
  - Managed by **L3 agent**
  - Name: **qrouter-**<*router_UUID*>
- DHCP namespace:
  - Every subnet **with DHCP enabled** has its own qdhcp namespace
  - Managed by **DHCP agent**
  - Name: **qdhcp-**<*network_UUID*>



The **ip netns list** command displays the defined namespaces.
The qrouter namespace:
- Represents the virtual router (router1 in this example)
- Is responsible for routing traffic to and from the virtual machine instances.
- Created when you set the **gateway interface** to connect the virtual router to an external (public) network.

The qdhcp namespace:
- Provides IP addresses to the virtual machines.
- Listens for DHCP requests.
- Created when you add an interface to connect a private subnetwork to the virtual router.

You can execute commands in a namespace:
ip netns exec <*namespace*> <*command*>
Several examples of executing commands in a namespace are discussed later in this lecture.

Namespaces are enabled in the neutron.conf configuration file (on the compute and network nodes):
allow_overlapping_ips = True

Issuing commands in namespaces

```
sudo ip netns exec qrouter-2ebce595-6c09-4306-af8f-be067f5be2b7 route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use  Iface
0.0.0.0         172.24.4.1      0.0.0.0         UG    0      0        0    qg-0c3b6d44-b7
10.0.0.0        0.0.0.0         255.255.255.192 U     0      0        0    qr-19c1c2f8-94
172.24.4.0      0.0.0.0         255.255.255.0   U     0      0        0    qg-0c3b6d44-b7
```

route table for qrouter namespace
qg-: external network
qr-: internal network

```
route -n
Kernel IP routing table
Destination     Gateway         Genmask         Flags Metric Ref    Use  Iface
0.0.0.0         172.31.16.1     0.0.0.0         UG    100    0        0    eth0
172.24.4.0      0.0.0.0         255.255.255.0   U     0      0        0    br-ex
172.31.16.0     0.0.0.0         255.255.240.0   U     0      0        0    eth0
172.31.16.1     0.0.0.0         255.255.255.255 UH    100    0        0    eth0
192.168.0.0     0.0.0.0         255.255.255.0   U     0      0        0    o-hm0
192.168.122.0   0.0.0.0         255.255.255.0   U     0      0        0    virbr0
```

route table for global namespace

The **route –n** command at the top displays the route table for the **qrouter** namespace. It connects the external (172) and internal (10) networks.
The **route –n** command at the bottom displays the route table for the global namespace on the network node. It connects to the management network (172).

Issuing commands in namespaces

```
sudo ip netns exec qrouter-2ebce595-6c09-4306-af8f-be067f5be2b7 ifconfig

qg-0c3b6d44-b7: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
     inet 172.24.4.19  netmask 255.255.255.0  broadcast 172.24.4.255
     inet6 fe80::f816:3eff:fe7d:31a2  prefixlen 64  scopeid 0x20<link>
     ether fa:16:3e:7d:31:a2  txqueuelen 1000  (Ethernet)
     RX packets 236  bytes 16325 (16.3 KB)
     RX errors 0  dropped 0  overruns 0  frame 0
     TX packets 282  bytes 20694 (20.6 KB)
     TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
qr-19c1c2f8-94: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
     inet 10.0.0.1  netmask 255.255.255.192  broadcast 10.0.0.63
     inet6 fe80::f816:3eff:febc:18b2  prefixlen 64  scopeid 0x20<link>
     ether fa:16:3e:bc:18:b2  txqueuelen 1000  (Ethernet)
     RX packets 3159  bytes 241119 (241.1 KB)
     RX errors 0  dropped 0  overruns 0  frame 0
     TX packets 1462  bytes 196804 (196.8 KB)
     TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

***qg-** interface:*
Port (1st available IP address in subnet pool) on external (public) network

***qr-** interface:*
Port (IP address of the gateway) on internal (private) network

The **route –n** command at the top displays the route table for the **qrouter** namespace. It connects the external (172) and internal (10) networks.
Another command example.
The **qr-** interface is assigned the IP address of the gateway (defined when the private subnet is created),10.0.0.1.
The **qg-** interface is assigned the first IP address in the subnet pool (defined when the public subnet is created), 172.24.4.19.

Octavia LBaaS v2

Load Balancer as a Service

Octavia is the OpenStack program that provides load balancing as a service (LBaaS). In particular, Octavia supports LBaaS v2, using HAproxy.
Octavia grew from the Neutron program and is included in this module because LBaaS is a network function.
The Octavia version for the OpenStack Rocky release is v0.9. The *latest* version is 4.0 with updates not covered in this lesson.

Octavia

- Application delivery, scaling, and availability are considered vital features of any cloud
- Load balancing is essential for enabling the applications
- Octavia provides a *reference implementation* of OpenStack LBaaS v2
  - "Cloud-like" load balancing operation
  - Manage virtual machines to provide load balancing services

MIRANTIS 31

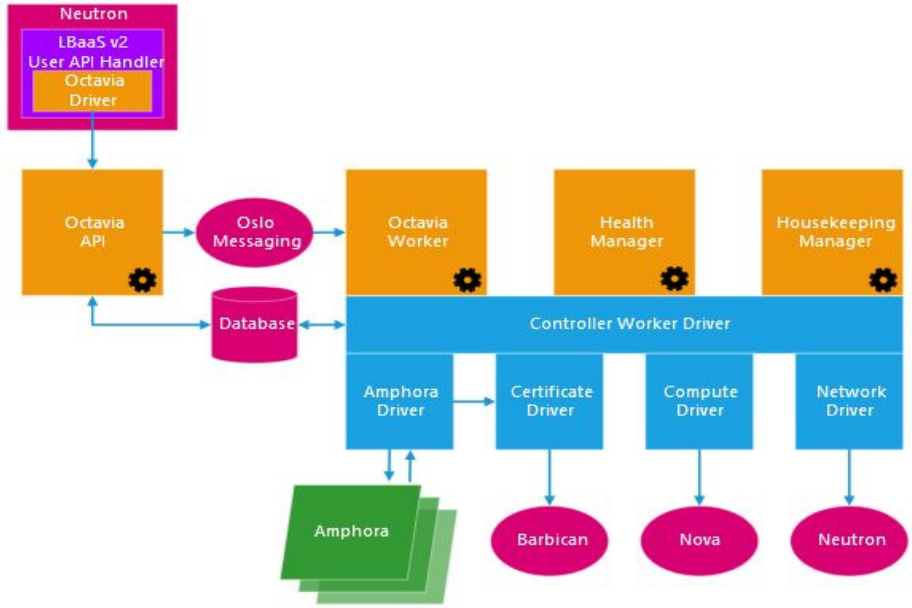
Octavia is an open source, operator-scale load balancing solution designed to work with OpenStack.

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. 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 servers—collectively 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."

Source:
https://docs.openstack.org/developer/octavia/main/introduction.html

Octavia (v0.9) components



MIRANTIS

32

This diagram is a simplified view of how the Octavia (version 0.9) components interact with OpenStack components to provide load balancer services. The various drivers (amphora, certificate, compute, and network) allow Octavia to be flexible in the choice of back-end plug-ins. OpenStack users interact with openstack CLI client to perform Octavia LBaaS v2 tasks.

The controller(s) and amphorae communicate over a LB network. The LB network is a traditional Neutron network to which both the controller and amphorae have access, but is not associated with any one project. The LB Network is also *not* part of the undercloud and should not be directly exposed to any OpenStack core components other than the Octavia Controller

Source:
https://docs.openstack.org/octavia/rocky/reference/introduction.html

In-depth source:
https://docs.openstack.org/developer/octavia/design/version0.5/component-design.html

Note: There are later versions of Octavia. v0.9 is the OpenStack Rocky release.

LBaaS amphorae

- Individual virtual machines, containers, or bare metal servers
  - Created dynamically when you create a load balancer
  - Default is 1 amphora per load balancer; supports a pool
- Implements load balancing services to tenant application environments
- *Reference implementation*
  - Image: Ubuntu 16.04 LTS VM running **HAProxy**
  - Uses **m1.amphora** flavor: 1 vCPU, 1GB RAM, 2GB disk
  - IP address in load balancer network, **lb-mgmt-net**
  - IP address in tenant (project) network
    - To reach back-end pool members, depending on how any given load balancing service is deployed by the tenant

Each amphora is a VM instance (can also be a container or bare metal server) requiring its own dedicated IP address in the LB network and the tenant network (owned by the project). The LB network has a **qdhcp namespace**, supporting the creation of the amphorae. Octavia manages load balancers with a pre-built Ubuntu 16.04 LTS image which contains:

- the amphora agent
- a load balancing application
- seeded with cryptographic certificates through the config drive at startup

Note: the amphora use a flavor that yields a small VM. Depending upon your environment, you might need larger flavors.

The amphorae communicate with the controllers in a “trusted” way, using certificates. As a result, users should not have command-line access to the amphorae. There is no need for it. **The amphorae should be a *black box* from the users’ perspective.**

The individual amphora and the load balancer network, lb-mgmt-net, are owned by the admin project. Only users with the **admin role** can see the amphorae, for example. Other possible roles are discussed later in this lesson.

The amphorae are monitored by sending heartbeat requests (over UDP) to the controller.

LBaaS controller

- Controller consists of 4 sub-components (daemons)
  - **API controller:** Processes API requests, passing to Octavia worker
  - **Octavia worker:** Fulfills requests
  - **Health manager:** Monitors amphorae to ensure they are up and running, and otherwise healthy
    - Handles failover events if amphorae fail unexpectedly
  - **Housekeeping manager:** Cleans up stale (deleted) database records, manages the spares pool, and manages amphora certificate rotation
- Communicates with amphorae using internal REST API
  - Requires additional TLS certificates

The controller API daemon must have access to both the LB Network and OpenStack components to coordinate and manage the overall activity of the Octavia load balancing system.
The health manager connects to the load balancer network, for example:

```
ifconfig
…
o-hm0: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1450
       inet 192.168.0.18  netmask 255.255.255.0  broadcast 192.168.0.255
       inet6 fe80::f816:3eff:fe3e:c373  prefixlen 64  scopeid 0x20<link>
       ether fa:16:3e:3e:c3:73  txqueuelen 1000  (Ethernet)
       RX packets 19698  bytes 4861451 (4.8 MB)
       RX errors 0  dropped 0  overruns 0  frame 0
       TX packets 5492  bytes 504757 (504.7 KB)
       TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

Octavia Features

- Elastic load balancing
  - Scale up and down with workload using alarms
- Active / Standby load balancer amphorae instances
- Users can define additional layer 7 (L7) policies and rules, for example:
  - any client request that matches the L7 rule: “request URI starts with ‘/api’ ” - should be routed to the “api” pool

Elastic load balancing is the main attraction of Octavia, and is also what enables enterprise use cases.
Furthermore, because the workload is handled by VMs, it can provide high availability features via active/standby amphorae instances.

Example: creating LBaaS topology

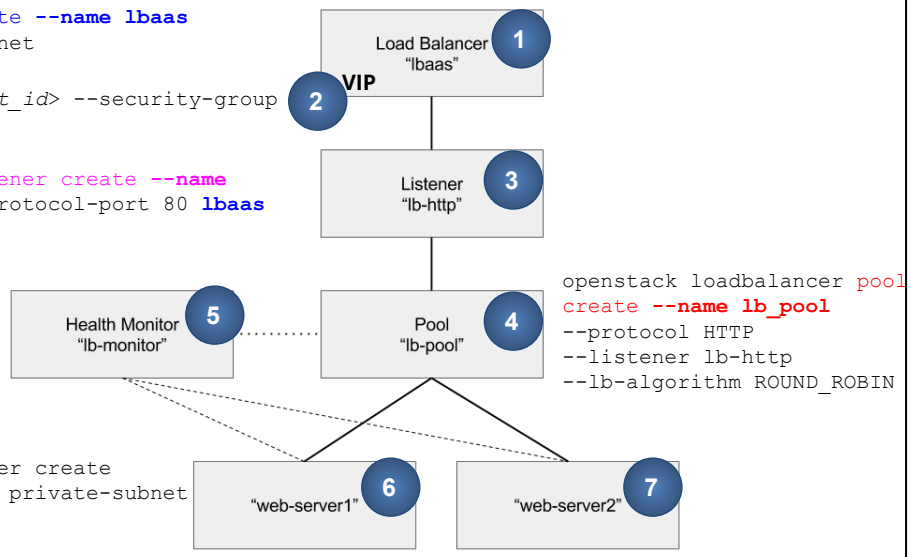
```
openstack loadbalancer create --name lbaas
--vip-subnet-id private-subnet

openstack port set <vip_port_id> --security-group
lb-sec

openstack loadbalancer listener create --name
lb-http --protocol HTTP --protocol-port 80 lbaas

openstack loadbalancer
healthmonitor create
--name lb-monitor --delay 2
--max-retries 2
--timeout 10 --type PING
lb_pool

openstack loadbalancer member create
--name member-N --subnet-id private-subnet
--address <web-server-N IP>
--protocol-port 80 lb-pool
```




```
openstack loadbalancer pool
create --name lb_pool
--protocol HTTP
--listener lb-http
--lb-algorithm ROUND_ROBIN
```


This slide shows the steps for creating a load balancer to front-end 2 HTTP (web) servers: web-server1 and web-server2. The web servers can be created at any point in the process.

(1) Create the load balancer, name **lbaas**
  (a) As the load balancer is created,a virtual IP address (VIP) is assigned to the load balancer
  (b) One or more amphorae will be provisioned at this time

(2) Assign security group rules to the VIP port of the load balancer. In this case, the security group, **lb-sec**, was created before this example.
  (a) Allow ingress traffic on port 80
  (b) Allow ingress traffic on port 22

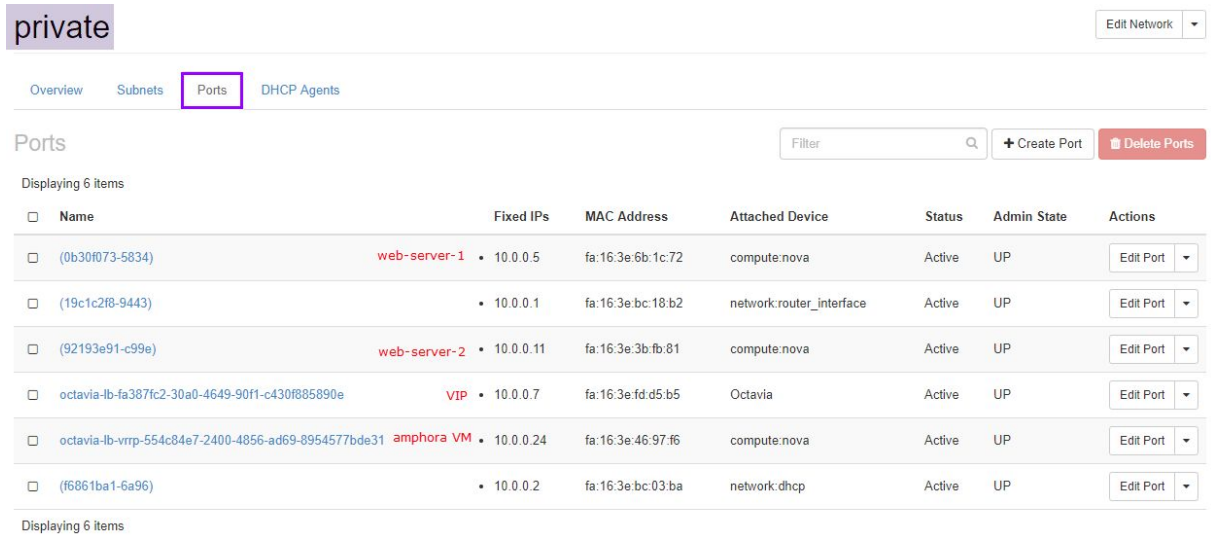
(3) Create a *listener* for the load balancer, name **lb-http**, for HTTP requests on port 80

(4) Create a load balancer *pool*, associated with the lb-http listener.
  (a) Pool members will be the web servers.
  (b) The algorithm used to distribute traffic will be round robin. Load balancing can also be done based on the source IP ofthe request or least connections.

(5) (Optional) Create a health monitor. In this case, **lb-monitor**, will *ping the pool members every 2 seconds*. After 2 failed responses, the member is removed from the pool.

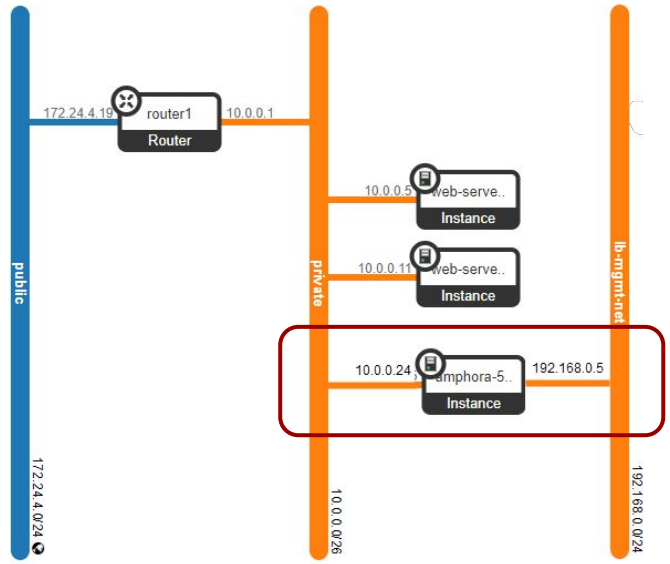
(6) Create *members* of the load balancer pool. The members are the web server instances. In this example, there are 2 web servers.
  (a) A common approach is to include autoscaling of the web servers,dynamically adding them to the load balancer pool as they are created.
  (b) This is typically implemented using a Heat template to create the load balancer resources, as well as the autoscaling resources.

Example: Private network ports



Using the previous **lbaas load balancer** example, this slide shows the ports allocated on the private network. Notice the ports for the load balancer VIP and amphora VM.

Example: Network topology



From the Dashboard UI, this slide shows an example network topology with the public (external), private (internal), and lb-mgmt-net (load balancer) networks. The web server instances are deployed to the private network. The amphora instance for the load balancer is connected to both the private and load balancer networks.

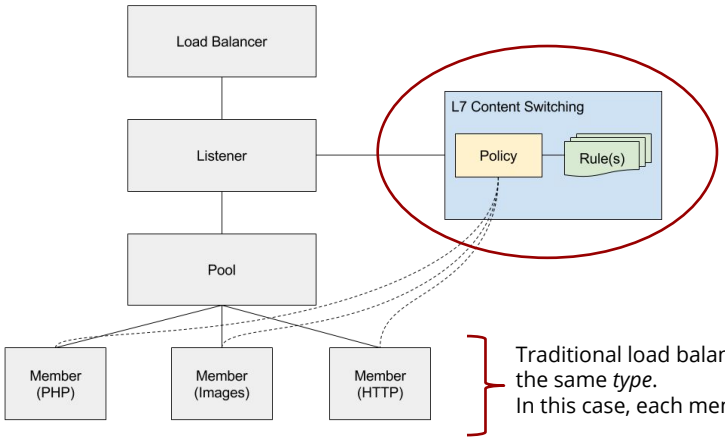
You must be a member of the admin project (with admin role) to see the amphora instance and lb-mgmt-net resources.

Note: this topology merges 2 different topology screens. It was created for lecture purposes.

Security groups

- Octavia requires several ports to be open
- Using devstack, there are 2 security groups:
  - lb-mgmt-sec-grp
    - Ingress ICMP
    - Ingress SSH (port 22)
    - Ingress TCP port 9443
  - lb-health-mgr-sec-grp
    - Ingress UDP port 5555

L7 content switching



MIRANTIS

40

L7 content switching is a type of load balancing done by setting rules and policies for an incoming request from an application. It is called L7 switching because the device switches requests based on the layer 7 (application) data.

In this simplified diagram, requests that come through the load balancer are forwarded to a L7 content switching server. Here, the contents of the request can be analyzed and compared with policies and rules. A policy is a collection of rules ANDed together. If the application data matches all rules, then it matches the policy.
Policy also enacts the final action of the request in content switching from the following:

1) block the request
2) forward to another URL
3) send it to a particular server (Member)

Unlike traditional load balancer members which are all identical - this example demonstrates that each member is expected to serve different type of content. If a valid request to fetch images comes in, then requests that match such rules will be forwarded to the Member which is specialized in serving image files.

Source:
https://wiki.openstack.org/wiki/Neutron/LBaaS/l7

(Keystone) Policies - reference

Role	Access
admin	User has admin access to all APIs
load-balancer_observer	User has access to load-balancer read-only APIs
load-balancer_global_observer	User has access to load-balancer read-only APIs including resources owned by others
load-balancer_member	User has access to load-balancer read and write APIs
load-balancer_quota_admin	User is considered an admin for quota APIs only
load-balancer_admin	User is considered an admin for all load-balancer APIs including resources owned by others

Users must have one of the following roles to access the load-balancer API (commands)

Lab exercises

Chapter 6: Load Balancer as a Service (Octavia)

Chapter 7: The Networking service (Neutron)