OpenStack

Cloud Administrator Guide

current (April 26, 2014)

BUILT FOR openstack™ CLOUD SOFTWARE

openstack™

docs.openstack.org

OpenStack Cloud Administrator Guide

current (2014-04-26)

OpenStack offers open source software for cloud administrators to manage and troubleshoot an OpenStack cloud.

Table of Contents

List of Figures

List of Tables

Preface

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take three forms:

Note

The information in a note is usually in the form of a handy tip or reminder.

Important

The information in an important notice is something you must be aware of before proceeding.

Warning

The information in warnings is critical. Warnings provide additional information about risk of data loss or security issues.

Command prompts

Commands prefixed with the # prompt are to be executed by the `root` user. These examples can also be executed by using the **sudo** command, if available.

Commands prefixed with the $ prompt can be executed by any user, including `root`.

Document change history

This version of the guide replaces and obsoletes all previous versions. The following table describes the most recent changes:

Revision Date	Summary of Changes
November 12, 2013	• Adds options for tuning operational status synchronization in the NSX plug-in.
October 17, 2013	• Havana release.
September 5, 2013	• Moves object storage monitoring section to this guide. • Removes redundant object storage information.
September 3, 2013	• Moved all but configuration and installation information from these component guides to create the new guide: • OpenStack Compute Administration Guide • OpenStack Networking Administration Guide • OpenStack Object Storage Administration Guide • OpenStack Block Storage Service Administration Guide

1. Get started with OpenStack

Table of Contents

The OpenStack project is an open source cloud computing platform for all types of clouds, which aims to be simple to implement, massively scalable, and feature rich. Developers and cloud computing technologists from around the world create the OpenStack project.

OpenStack provides an Infrastructure-as-a-Service (*IaaS*) solution through a set of interrelated services. Each service offers an application programming interface (*API*) that facilitates this integration. Depending on your needs, you can install some or all services.

The following table describes the OpenStack services that make up the OpenStack architecture:

Table 1.1. OpenStack services

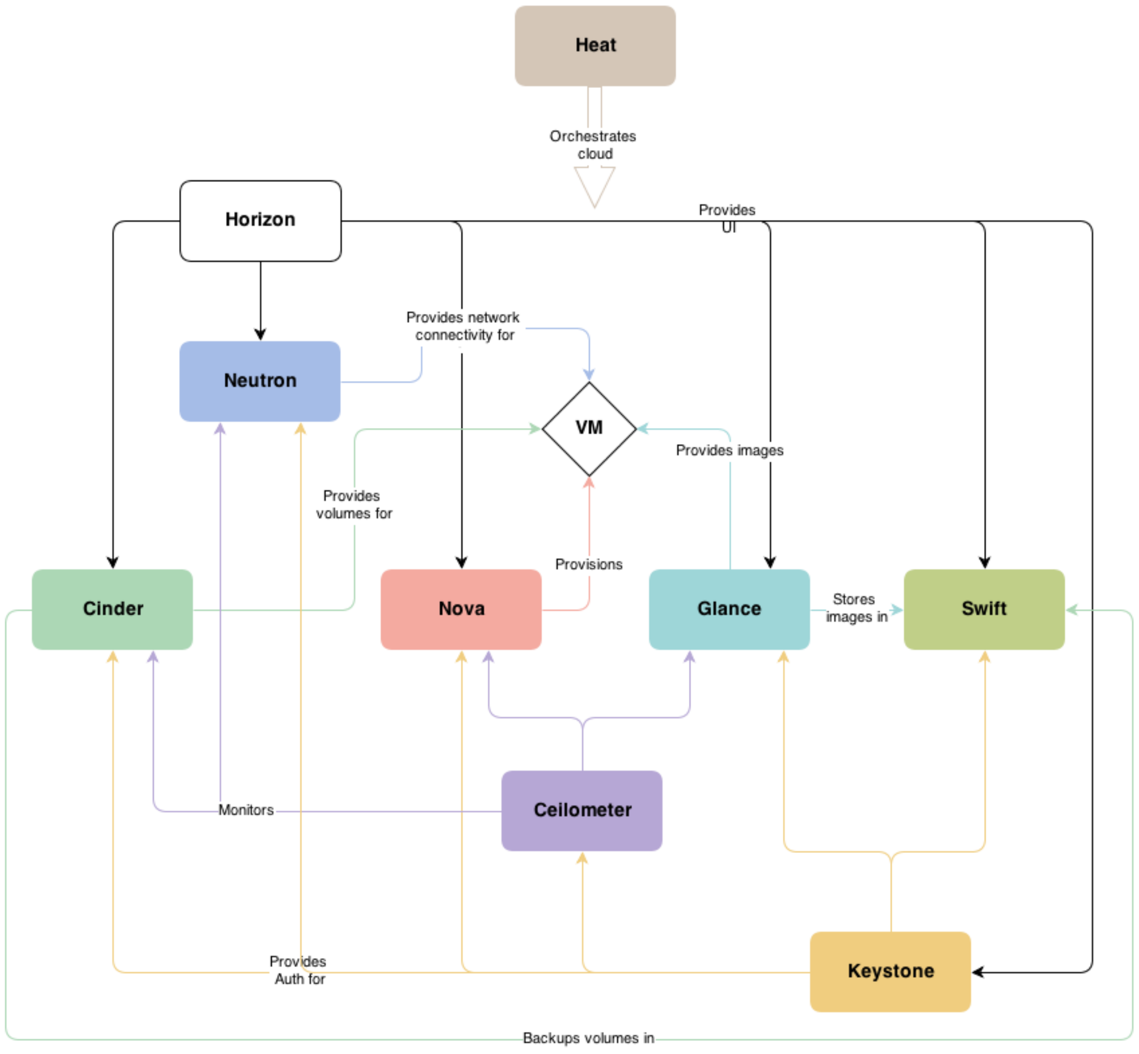
Service	Project name	Description
Dashboard	*Horizon*	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	*Nova*	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decomissioning of virtual machines on demand.
Networking	*Neutron*	Enables network connectivity as a service for other OpenStack services, such as OpenStack Compute. Provides an API for users to define networks and the attachments into them. Has a pluggable architecture that supports many popular networking vendors and technologies.
	Storage	
Object Storage	*Swift*	Stores and retrieves arbitrary unstructured data objects via a *RESTful*, HTTP based API. It is highly fault tolerant with its data replication and scale out architecture. Its implementation is not like a file server with mountable directories.
Block Storage	*Cinder*	Provides persistent block storage to running instances. Its pluggable driver architecture facilitates the creation and management of block storage devices.
	Shared services	
Identity service	*Keystone*	Provides an authentication and authorization service for other OpenStack services. Provides a catalog of endpoints for all OpenStack services.
Image Service	*Glance*	Stores and retrieves virtual machine disk images. OpenStack Compute makes use of this during instance provisioning.
Telemetry	*Ceilometer*	Monitors and meters the OpenStack cloud for billing, benchmarking, scalability, and statistical purposes.
	Higher-level services	

Service	Project name	Description
Orchestration	*Heat*	Orchestrates multiple composite cloud applications by using either the native *HOT* template format or the AWS CloudFormation template format, through both an OpenStack-native REST API and a CloudFormation-compatible Query API.
Database Service	*Trove*	Provides scalable and reliable Cloud Database-as-a-Service functionality for both relational and non-relational database engines.

Conceptual architecture

The following diagram shows the relationships among the OpenStack services:

Figure 1.1. OpenStack conceptual architecture



Logical architecture

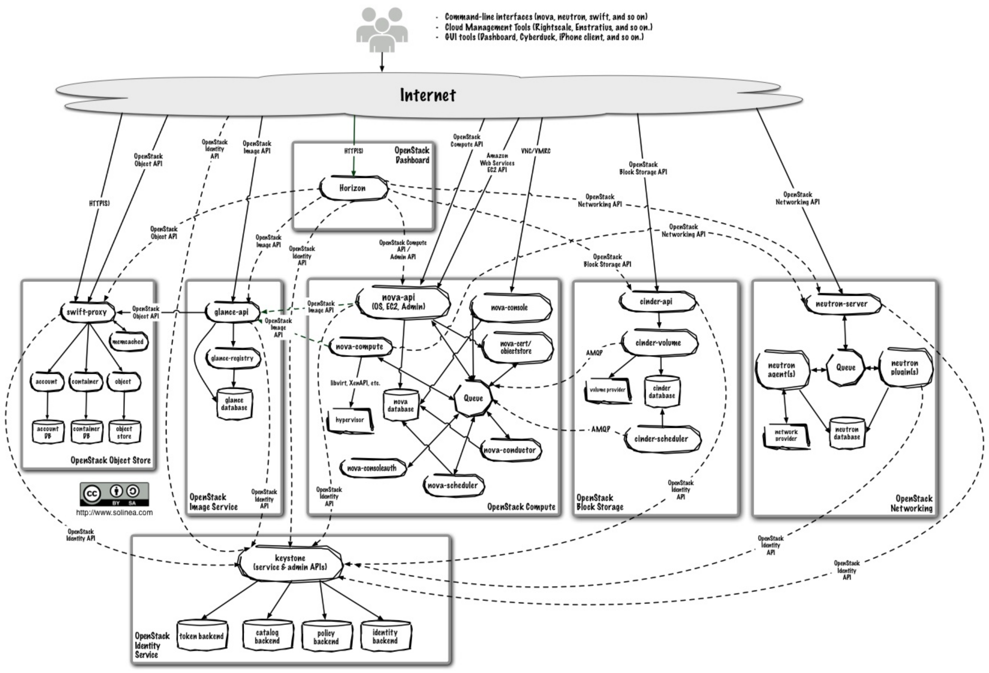
To design, deploy, and configure OpenStack, administrators must understand the logical architecture.

OpenStack modules are one of the following types:

Daemon	Runs as a background process. On Linux platforms, a daemon is usually installed as a service.
Script	Installs a virtual environment and runs tests. For example, the `run_tests.sh` script installs a virtual environment and runs unit tests on a service.
Command-line interface (CLI)	Enables users to submit API calls to OpenStack services through easy-to-use commands.

The following diagram shows the most common, but not the only, architecture for an OpenStack cloud:

Figure 1.2. Logical architecture



As in Figure 1.1, "OpenStack conceptual architecture" [2], end users can interact through the dashboard, CLIs, and APIs. All services authenticate through a common Identity Service and individual services interact with each other through public APIs, except where privileged administrator commands are necessary.

OpenStack services

This section describes OpenStack services in detail.

Compute service

The Compute service is a cloud computing fabric controller, which is the main part of an IaaS system. Use it to host and manage cloud computing systems. The main modules are implemented in Python.

Compute interacts with the Identity Service for authentication, Image Service for images, and the Dashboard for the user and administrative interface. Access to images is limited by project and by user; quotas are limited per project (for example, the number of instances). The Compute service scales horizontally on standard hardware, and downloads images to launch instances as required.

The Compute service is made up of the following functional areas and their underlying components:

API

- `nova-api` service. Accepts and responds to end user compute API calls. Supports the OpenStack Compute API, the Amazon EC2 API, and a special Admin API for privileged users to perform administrative actions. Also, initiates most orchestration activities, such as running an instance, and enforces some policies.
- `nova-api-metadata` service. Accepts metadata requests from instances. The `nova-api-metadata` service is generally only used when you run in multi-host mode with `nova-network` installations. For details, see Metadata service in the *Cloud Administrator Guide*.

  On Debian systems, it is included in the `nova-api` package, and can be selected through debconf.

Compute core

- `nova-compute` process. A worker daemon that creates and terminates virtual machine instances through hypervisor APIs. For example, XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware, and so on. The process by which it does so is fairly complex but the basics are simple: Accept actions from the queue and perform a series of system commands, like launching a KVM instance, to carry them out while updating state in the database.
- `nova-scheduler` process. Conceptually the simplest piece of code in Compute. Takes a virtual machine instance request from the queue and determines on which compute server host it should run.
- `nova-conductor` module. Mediates interactions between `nova-compute` and the database. Aims to eliminate direct accesses to the cloud database made by `nova-compute`. The `nova-conductor` module scales horizontally. However, do not deploy it on any nodes where `nova-compute` runs. For more information, see A new Nova service: nova-conductor.

Networking for VMs

- `nova-network` worker daemon. Similar to `nova-compute`, it accepts networking tasks from the queue and performs tasks to manipulate the network, such as setting up bridging interfaces or changing iptables rules. This functionality is being migrated to OpenStack Networking, which is a separate OpenStack service.

- `nova-dhcpbridge` script. Tracks IP address leases and records them in the database by using the dnsmasq `dhcp-script` facility. This functionality is being migrated to OpenStack Networking. OpenStack Networking provides a different script.

Console interface

- `nova-consoleauth` daemon. Authorizes tokens for users that console proxies provide. See `nova-novncproxy` and `nova-xvpnvcproxy`. This service must be running for console proxies to work. Many proxies of either type can be run against a single `nova-consoleauth` service in a cluster configuration. For information, see About nova-consoleauth.

- `nova-novncproxy` daemon. Provides a proxy for accessing running instances through a VNC connection. Supports browser-based novnc clients.

- `nova-xvpnvncproxy` daemon. A proxy for accessing running instances through a VNC connection. Supports a Java client specifically designed for OpenStack.

- `nova-cert` daemon. Manages x509 certificates.

In Debian, a unique nova-consoleproxy package provides the nova-novncproxy, nova-spicehtml5proxy, and nova-xvpvncproxy packages. To select packages, edit the `/etc/default/nova-consoleproxy` file or use the debconf interface. You can also manually edit the `/etc/default/nova-consoleproxy` file and stop and start the console daemons.

Image management (EC2 scenario)

- `nova-objectstore` daemon. Provides an S3 interface for registering images with the Image Service. Mainly used for installations that must support euca2ools. The euca2ools tools talk to `nova-objectstore` in *S3 language*, and `nova-objectstore` translates S3 requests into Image Service requests.

- euca2ools client. A set of command-line interpreter commands for managing cloud resources. Though not an OpenStack module, you can configure `nova-api` to support this EC2 interface. For more information, see the Eucalyptus 3.4 Documentation.

Command-line clients and other interfaces

- nova client. Enables users to submit commands as a tenant administrator or end user.

- nova-manage client. Enables cloud administrators to submit commands.

Other components

- The queue. A central hub for passing messages between daemons. Usually implemented with RabbitMQ, but could be any AMQP message queue, such as Apache Qpid or Zero MQ.

- SQL database. Stores most build-time and runtime states for a cloud infrastructure. Includes instance types that are available for use, instances in use, available networks, and projects. Theoretically, OpenStack Compute can support any database that SQL-Alchemy supports, but the only databases widely used are SQLite3 databases (only appropriate for test and development work), MySQL, and PostgreSQL.

The Compute service interacts with other OpenStack services: Identity Service for authentication, Image Service for images, and the OpenStack dashboard for a web interface.

Storage concepts

The OpenStack stack uses the following storage types:

Table 1.2. Storage types

On-instance / ephemeral	Block storage (Cinder)	Object Storage (Swift)
Runs operating systems and provides scratch space	Used for adding additional persistent storage to a virtual machine (VM)	Used for storing virtual machine images and data
Persists until VM is terminated	Persists until deleted	Persists until deleted
Access associated with a VM	Access associated with a VM	Available from anywhere
Implemented as a filesystem underlying OpenStack Compute	Mounted via OpenStack Block-Storage controlled protocol (for example, iSCSI)	REST API
Administrator configures size setting, based on flavors	Sizings based on need	Easily scalable for future growth
Example: 10 GB first disk, 30 GB/core second disk	Example: 1 TB "extra hard drive"	Example: 10s of TBs of data set storage

Other points of note include:

- *OpenStack Object Storage is not used like a traditional hard drive.* Object storage is all about relaxing some of the constraints of a POSIX-style file system. The access to it is API-based (and the API uses http). This is a good idea as if you don't have to provide atomic operations (that is, you can rely on eventual consistency), you can much more easily scale a storage system and avoid a central point of failure.
- *The OpenStack Image Service is used to manage the virtual machine images in an OpenStack cluster, not store them.* Instead, it provides an abstraction to different methods for storage - a bridge to the storage, not the storage itself.
- *OpenStack Object Storage can function on its own.* The Object Storage (swift) product can be used independently of the Compute (nova) product.

Object Storage service

The Object Storage service is a highly scalable and durable multi-tenant object storage system for large amounts of unstructured data at low cost through a RESTful HTTP API.

It includes the following components:

- Proxy servers (`swift-proxy-server`). Accepts Object Storage API and raw HTTP requests to upload files, modify metadata, and create containers. It also serves file or container listings to web browsers. To improve performance, the proxy server can use an optional cache usually deployed with memcache.
- Account servers (`swift-account-server`). Manage accounts defined with the Object Storage service.
- Container servers (`swift-container-server`). Manage a mapping of containers, or folders, within the Object Storage service.
- Object servers (`swift-object-server`). Manage actual objects, such as files, on the storage nodes.
- A number of periodic processes. Performs housekeeping tasks on the large data store. The replication services ensure consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.
- Configurable WSGI middleware that handles authentication. Usually the Identity Service.

Block Storage

The Block Storage service enables management of volumes, volume snapshots, and volume types. It includes the following components:

- `cinder-api`: Accepts API requests and routes them to `cinder-volume` for action.
- `cinder-volume`: Responds to requests to read from and write to the Block Storage database to maintain state, interacting with other processes (like `cinder-scheduler`) through a message queue and directly upon block storage providing hardware or software. It can interact with a variety of storage providers through a driver architecture.

- `cinder-scheduler` daemon: Like the `nova-scheduler`, picks the optimal block storage provider node on which to create the volume.
- Messaging queue: Routes information between the Block Storage service processes.

The Block Storage service interacts with Compute to provide volumes for instances.

Networking service overview

Provides network-connectivity-as-a-service between interface devices that are managed by other OpenStack services, usually Compute. Enables users to create and attach interfaces to networks. Like many OpenStack services, OpenStack Networking is highly configurable due to its plug-in architecture. These plug-ins accommodate different networking equipment and software. Consequently, the architecture and deployment vary dramatically.

Includes the following components:

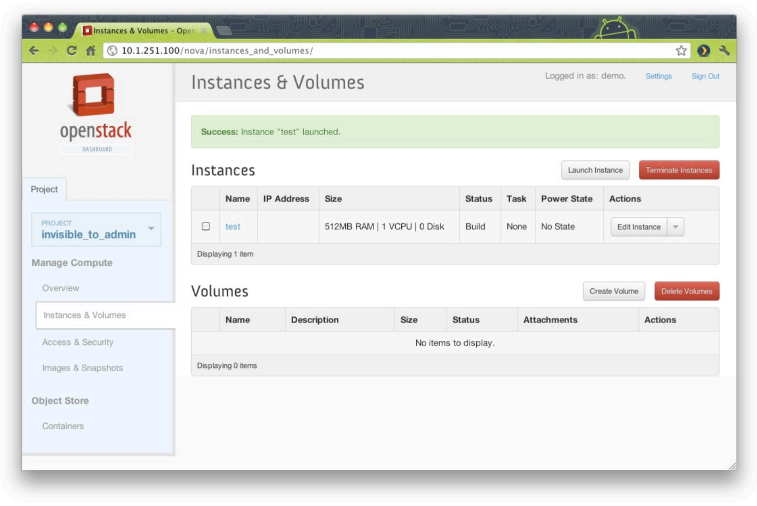
- `neutron-server`. Accepts and routes API requests to the appropriate OpenStack Networking plug-in for action.
- OpenStack Networking plug-ins and agents. Plugs and unplugs ports, creates networks or subnets, and provides IP addressing. These plug-ins and agents differ depending on the vendor and technologies used in the particular cloud. OpenStack Networking ships with plug-ins and agents for Cisco virtual and physical switches, NEC OpenFlow products, Open vSwitch, Linux bridging, Ryu Network Operating System, and the VMware NSX product.

  The common agents are L3 (layer 3), DHCP (dynamic host IP addressing), and a plug-in agent.
- Messaging queue. Most OpenStack Networking installations make use of a messaging queue to route information between the neutron-server and various agents as well as a database to store networking state for particular plug-ins.

OpenStack Networking interacts mainly with OpenStack Compute, where it provides networks and connectivity for its instances.

Dashboard

The dashboard is a modular Django web application that provides a graphical interface to OpenStack services.



Success: Instance "test" launched.

Instances

	Name	IP Address	Size	Status	Task	Power State	Actions
☐	test		512MB RAM | 1 VCPU | 0 Disk	Build	None	No State	Edit Instance

Displaying 1 item

Volumes

	Name	Description	Size	Status	Attachments	Actions
			No items to display.			

Displaying 0 items

The dashboard is usually deployed through mod_wsgi in Apache. You can modify the dashboard code to make it suitable for different sites.

From a network architecture point of view, this service must be accessible to customers and the public API for each OpenStack service. To use the administrator functionality for other services, it must also connect to Admin API endpoints, which should not be accessible by customers.

Identity Service concepts

The *Identity Service* performs the following functions:

- User management. Tracks users and their permissions.
- *Service catalog*. Provides a catalog of available services with their API endpoints.

To understand the Identity Service, you must understand the following concepts:

User	Digital representation of a person, system, or service who uses OpenStack cloud services. The Identity Service validates that incoming requests are made by the user who claims to be making the call. Users have a login and may be assigned tokens to access resources. Users can be directly assigned to a particular tenant and behave as if they are contained in that tenant.
Credentials	Data that is known only by a user that proves who they are. In the Identity Service, examples are: User name and password, user name and API key, or an authentication token provided by the Identity Service.

Authentication	The act of confirming the identity of a user. The Identity Service confirms an incoming request by validating a set of credentials supplied by the user. These credentials are initially a user name and password or a user name and API key. In response to these credentials, the Identity Service issues an authentication token to the user, which the user provides in subsequent requests.
Token	An arbitrary bit of text that is used to access resources. Each token has a scope which describes which resources are accessible with it. A token may be revoked at any time and is valid for a finite duration. While the Identity Service supports token-based authentication in this release, the intention is for it to support additional protocols in the future. The intent is for it to be an integration service foremost, and not aspire to be a full-fledged identity store and management solution.
Tenant	A container used to group or isolate resources and/or identity objects. Depending on the service operator, a tenant may map to a customer, account, organization, or project.
Service	An OpenStack service, such as Compute (Nova), Object Storage (Swift), or Image Service (Glance). Provides one or more endpoints through which users can access resources and perform operations.
Endpoint	A network-accessible address, usually described by a URL, from where you access a service. If using an extension for templates, you can create an endpoint template, which represents the templates of all the consumable services that are available across the regions.
Role	A personality that a user assumes that enables them to perform a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges. In the Identity Service, a token that is issued to a user includes the list of roles that user has. Services that are being called by that user determine how they interpret the set of roles a user has and to which operations or resources each role grants access.

The following diagram shows the Identity Service process flow:

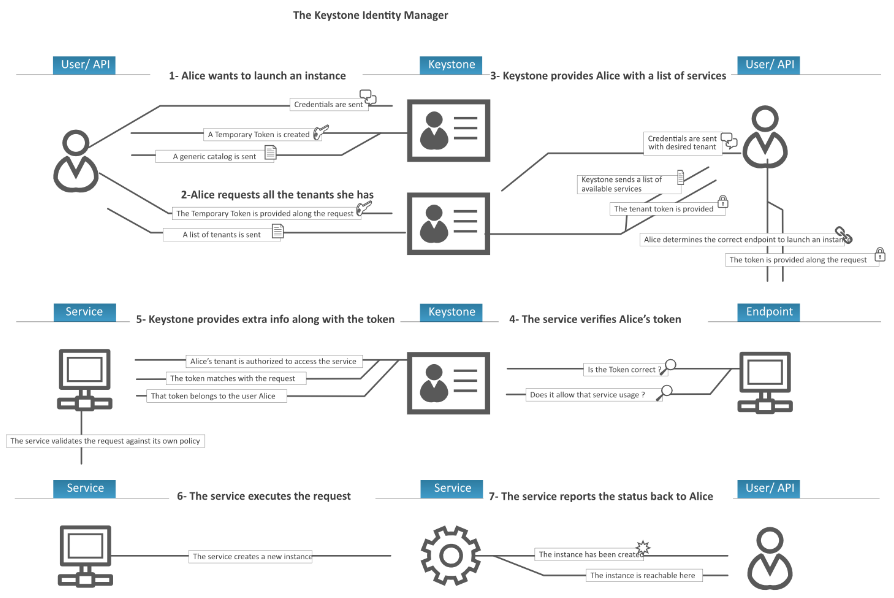

The Keystone Identity Manager
User/ API
1- Alice wants to launch an instance
Keystone
3- Keystone provides Alice with a list of services
User/ API
Credentials are sent
A Temporary Token is created
A generic catalog is sent
2-Alice requests all the tenants she has
The Temporary Token is provided along the request
A list of tenants is sent
Credentials are sent with desired tenant
Keystone sends a list of available services
The tenant token is provided
Alice determines the correct endpoint to launch an instance
The token is provided along the request
Service
5- Keystone provides extra info along with the token
Keystone
4- The service verifies Alice's token
Endpoint
Alice's tenant is authorized to access the service
The token matches with the request
That token belongs to the user Alice
Is the Token correct ?
Does it allow that service usage ?
The service validates the request against its own policy
Service
6- The service executes the request
Service
7- The service reports the status back to Alice
User/ API
The service creates a new instance
The instance has been created
The instance is reachable here

Image Service overview

The Image Service includes the following components:

- `glance-api`. Accepts Image API calls for image discovery, retrieval, and storage.
- `glance-registry`. Stores, processes, and retrieves metadata about images. Metadata includes items such as size and type.

Security note

The registry is a private internal service meant only for use by the Image Service itself. Do not expose it to users.

- Database. Stores image metadata. You can choose your database depending on your preference. Most deployments use MySQL or SQlite.
- Storage repository for image files. The Image Service supports a variety of repositories including normal file systems, Object Storage, RADOS block devices, HTTP, and Amazon S3. Some types of repositories support only read-only usage.

A number of periodic processes run on the Image Service to support caching. Replication services ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters, and reapers.

As shown in the section called "Conceptual architecture" [2], the Image Service is central to the overall IaaS picture. It accepts API requests for images or image metadata from end users or Compute components and can store its disk files in the Object Storage Service.

Telemetry

The Telemetry module:

- Efficiently collects the metering data about the CPU and network costs.
- Collects data by monitoring notifications sent from services or by polling the infrastructure.
- Configures the type of collected data to meet various operating requirements. Accessing and inserting the metering data through the REST API.
- Expands the framework to collect custom usage data by additional plug-ins.
- Produces signed metering messages that cannot be repudiated.

The system consists of the following basic components:

- A compute agent (`ceilometer-agent-compute`). Runs on each compute node and polls for resource utilization statistics. There may be other types of agents in the future, but for now we will focus on creating the compute agent.
- A central agent (`ceilometer-agent-central`). Runs on a central management server to poll for resource utilization statistics for resources not tied to instances or compute nodes.

- A collector (`ceilometer-collector`). Runs on one or more central management servers to monitor the message queues (for notifications and for metering data coming from the agent). Notification messages are processed and turned into metering messages and sent back out onto the message bus using the appropriate topic. Telemetry messages are written to the data store without modification.
- An alarm notifier (`ceilometer-alarm-notifier`). Runs on one or more central management servers to allow settting alarms based on threshold evaluation for a collection of samples.
- A data store. A database capable of handling concurrent writes (from one or more collector instances) and reads (from the API server).
- An API server (`ceilometer-api`). Runs on one or more central management servers to provide access to the data from the data store.

These services communicate by using the standard OpenStack messaging bus. Only the collector and API server have access to the data store.

Orchestration service overview

The Orchestration service provides a template-based orchestration for describing a cloud application by running OpenStack API calls to generate running cloud applications. The software integrates other core components of OpenStack into a one-file template system. The templates enable you to create most OpenStack resource types, such as instances, floating IPs, volumes, security groups, users, and so on. Also, provides some more advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. By providing very tight integration with other OpenStack core projects, all OpenStack core projects could receive a larger user base.

The service enables deployers to integrate with the Orchestration service directly or through custom plug-ins.

The Orchestration service consists of the following components:

- `heat` command-line client. A CLI that communicates with the heat-api to run AWS CloudFormation APIs. End developers could also use the Orchestration REST API directly.
- `heat-api` component. Provides an OpenStack-native REST API that processes API requests by sending them to the heat-engine over RPC.
- `heat-api-cfn` component. Provides an AWS Query API that is compatible with AWS CloudFormation and processes API requests by sending them to the heat-engine over RPC.
- `heat-engine`. Orchestrates the launching of templates and provides events back to the API consumer.

Feedback

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

2. Identity management

Table of Contents

The OpenStack Identity Service, code-named Keystone, is the default identity management system for OpenStack. After you install the Identity Service, you configure it through the `etc/keystone.conf` configuration file and, possibly, a separate logging configuration file. You initialize data into the Identity Service by using the **keystone** command-line client.

Identity Service concepts

User management

The main components of Identity user management are:

- **User**. Represents a human user. Has associated information such as user name, password, and email. This example creates a user named `alice`:

```
$ keystone user-create --name=alice --pass=mypassword123 --email=
alice@example.com
```

- **Tenant**. A project, group, or organization. When you make requests to OpenStack services, you must specify a tenant. For example, if you query the Compute service for a list of running instances, you get a list of all running instances in the tenant that you specified in your query. This example creates a tenant named `acme`:

```
$ keystone tenant-create --name=acme
```

Note

Because the term *project* was used instead of *tenant* in earlier versions of OpenStack Compute, some command-line tools use `--project_id` instead of `--tenant-id` or `--os-tenant-id` to refer to a tenant ID.

- **Role**. Captures the operations that a user can perform in a given tenant.

This example creates a role named `compute-user`:

```
$ keystone role-create --name=compute-user
```

Note

Individual services, such as Compute and the Image Service, assign meaning to roles. In the Identity Service, a role is simply a name.

The Identity Service assigns a tenant and a role to a user. You might assign the `compute-user` role to the `alice` user in the `acme` tenant:

```
$ keystone user-list
+--------+---------+-------------------+--------+
|   id   | enabled |       email       |  name  |
+--------+---------+-------------------+--------+
| 892585 |   True  | alice@example.com | alice  |
+--------+---------+-------------------+--------+
```

```
$ keystone role-list
+--------+--------------+
|   id   |     name     |
+--------+--------------+
| 9a764e | compute-user |
+--------+--------------+
```

```
$ keystone tenant-list
+--------+------+---------+
|   id   | name | enabled |
+--------+------+---------+
| 6b8fd2 | acme |   True  |
+--------+------+---------+
```

```
$ keystone user-role-add --user=892585 --role=9a764e --tenant-id=6b8fd2
```

A user can have different roles in different tenants. For example, Alice might also have the `admin` role in the `Cyberdyne` tenant. A user can also have multiple roles in the same tenant.

The `/etc/[SERVICE_CODENAME]/policy.json` file controls the tasks that users can perform for a given service. For example, `/etc/nova/policy.json` specifies the access policy for the Compute service, `/etc/glance/policy.json` specifies the access policy for the Image Service, and `/etc/keystone/policy.json` specifies the access policy for the Identity Service.

The default `policy.json` files in the Compute, Identity, and Image Service recognize only the `admin` role: all operations that do not require the `admin` role are accessible by any user that has any role in a tenant.

If you wish to restrict users from performing operations in, say, the Compute service, you need to create a role in the Identity Service and then modify `/etc/nova/policy.json` so that this role is required for Compute operations.

For example, this line in `/etc/nova/policy.json` specifies that there are no restrictions on which users can create volumes: if the user has any role in a tenant, they can create volumes in that tenant.

```
"volume:create": [],
```

To restrict creation of volumes to users who had the `compute-user` role in a particular tenant, you would add `"role:compute-user"`, like so:

```
"volume:create": ["role:compute-user"],
```

To restrict all Compute service requests to require this role, the resulting file would look like:

```
{
   "admin_or_owner":[
      [
         "role:admin"
      ],
      [
         "project_id:%(project_id)s"
      ]
   ],
   "default":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute:create":[
      "role:compute-user"
   ],
   "compute:create:attach_network":[
      "role:compute-user"
   ],
   "compute:create:attach_volume":[
      "role:compute-user"
   ],
   "compute:get_all":[
      "role:compute-user"
   ],
   "compute:unlock_override":[
      "rule:admin_api"
   ],
   "admin_api":[
      [
         "role:admin"
      ]
   ],
   "compute_extension:accounts":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:admin_actions":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:admin_actions:pause":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:unpause":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:suspend":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:resume":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:lock":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:unlock":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:resetNetwork":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:admin_actions:injectNetworkInfo":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:admin_actions:createBackup":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:admin_actions:migrateLive":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:admin_actions:migrate":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:aggregates":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:certificates":[
      "role:compute-user"
   ],
   "compute_extension:cloudpipe":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:console_output":[
      "role:compute-user"
   ],
   "compute_extension:consoles":[
      "role:compute-user"
   ],
   "compute_extension:createserverext":[
      "role:compute-user"
   ],
   "compute_extension:deferred_delete":[
      "role:compute-user"
   ],
   "compute_extension:disk_config":[
      "role:compute-user"
   ],
   "compute_extension:evacuate":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:extended_server_attributes":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:extended_status":[
      "role:compute-user"
   ],
   "compute_extension:flavorextradata":[
      "role:compute-user"
   ],
   "compute_extension:flavorextraspecs":[
      "role:compute-user"
   ],
   "compute_extension:flavormanage":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:floating_ip_dns":[
      "role:compute-user"
   ],
   "compute_extension:floating_ip_pools":[
      "role:compute-user"
   ],
   "compute_extension:floating_ips":[
      "role:compute-user"
   ],
   "compute_extension:hosts":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:keypairs":[
      "role:compute-user"
   ],
   "compute_extension:multinic":[
      "role:compute-user"
   ],
   "compute_extension:networks":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:quotas":[
      "role:compute-user"
   ],
   "compute_extension:rescue":[
      "role:compute-user"
   ],
   "compute_extension:security_groups":[
      "role:compute-user"
   ],
   "compute_extension:server_action_list":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:server_diagnostics":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:simple_tenant_usage:show":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "compute_extension:simple_tenant_usage:list":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:users":[
      [
         "rule:admin_api"
      ]
   ],
   "compute_extension:virtual_interfaces":[
      "role:compute-user"
   ],
   "compute_extension:virtual_storage_arrays":[
      "role:compute-user"
   ],
   "compute_extension:volumes":[
      "role:compute-user"
   ],
   "compute_extension:volume_attachments:index":[
      "role:compute-user"
   ],
   "compute_extension:volume_attachments:show":[
      "role:compute-user"
   ],
   "compute_extension:volume_attachments:create":[
      "role:compute-user"
   ],
   "compute_extension:volume_attachments:delete":[
      "role:compute-user"
   ],
   "compute_extension:volumetypes":[
      "role:compute-user"
   ],
   "volume:create":[
      "role:compute-user"
   ],
   "volume:get_all":[
      "role:compute-user"
   ],
   "volume:get_volume_metadata":[
      "role:compute-user"
   ],
   "volume:get_snapshot":[
      "role:compute-user"
   ],
   "volume:get_all_snapshots":[
      "role:compute-user"
   ],
   "network:get_all_networks":[
      "role:compute-user"
   ],
   "network:get_network":[
      "role:compute-user"
   ],
   "network:delete_network":[
      "role:compute-user"
   ],
   "network:disassociate_network":[
      "role:compute-user"
   ],
   "network:get_vifs_by_instance":[
      "role:compute-user"
   ],
   "network:allocate_for_instance":[
      "role:compute-user"
   ],
   "network:deallocate_for_instance":[
      "role:compute-user"
   ],
   "network:validate_networks":[
      "role:compute-user"
   ],
   "network:get_instance_uuids_by_ip_filter":[
      "role:compute-user"
   ],
   "network:get_floating_ip":[
      "role:compute-user"
   ],
   "network:get_floating_ip_pools":[
      "role:compute-user"
   ],
   "network:get_floating_ip_by_address":[
      "role:compute-user"
   ],
   "network:get_floating_ips_by_project":[
      "role:compute-user"
   ],
   "network:get_floating_ips_by_fixed_address":[
      "role:compute-user"
   ],
   "network:allocate_floating_ip":[
      "role:compute-user"
   ],
   "network:deallocate_floating_ip":[
      "role:compute-user"
   ],
   "network:associate_floating_ip":[
      "role:compute-user"
   ],
   "network:disassociate_floating_ip":[
      "role:compute-user"
   ],
   "network:get_fixed_ip":[
      "role:compute-user"
   ],
   "network:add_fixed_ip_to_instance":[
      "role:compute-user"
   ],
   "network:remove_fixed_ip_from_instance":[
      "role:compute-user"
   ],
   "network:add_network_to_project":[
      "role:compute-user"
   ],
   "network:get_instance_nw_info":[
      "role:compute-user"
   ],
   "network:get_dns_domains":[
      "role:compute-user"
   ],
   "network:add_dns_entry":[
      "role:compute-user"
   ],
   "network:modify_dns_entry":[
      "role:compute-user"
   ],
   "network:delete_dns_entry":[
      "role:compute-user"
   ],
   "network:get_dns_entries_by_address":[
      "role:compute-user"
   ],
   "network:get_dns_entries_by_name":[
      "role:compute-user"
   ],
   "network:create_private_dns_domain":[
      "role:compute-user"
   ],
   "network:create_public_dns_domain":[
      "role:compute-user"
   ],
   "network:delete_dns_domain":[
      "role:compute-user"
   ]
}
```

Service management

The Identity Service provides identity, token, catalog, and policy services. It consists of:

- `keystone-all`. Starts both the service and administrative APIs in a single process to provide Catalog, Authorization, and Authentication services for OpenStack.
- Identity Service functions. Each has a pluggable back end that allows different ways to use the particular service. Most support standard back ends like LDAP or SQL.

The Identity Service also maintains a user that corresponds to each service, such as, a user named *nova* for the Compute service, and a special service tenant called *service*.

For information about how to create services and endpoints, see the *OpenStack Admin User Guide*.

Groups

A group is a collection of users. Administrators can create groups and add users to them. Then, rather than assign a role to each user individually, assign a role to the group. Every group is in a domain. Groups were introduced with the Identity API v3.

Identity API V3 provides the following group-related operations:

- Create a group
- Delete a group
- Update a group (change its name or description)
- Add a user to a group
- Remove a user from a group
- List group members
- List groups for a user
- Assign a role on a tenant to a group
- Assign a role on a domain to a group
- Query role assignments to groups

Note

The Identity service server might not allow all operations. For example, if using the Identity server with the LDAP Identity back end and group updates are disabled, then a request to create, delete, or update a group fails.

Here are a couple of examples:

- Group A is granted Role A on Tenant A. If User A is a member of Group A, when User A gets a token scoped to Tenant A, the token also includes Role A.
- Group B is granted Role B on Domain B. If User B is a member of Domain B, if User B gets a token scoped to Domain B, the token also includes Role B.

Domains

A domain defines administrative boundaries for the management of Identity entities. A domain may represent an individual, company, or operator-owned space. It is used for exposing administrative activities directly to the system users.

A domain is a collection of tenants, users, and roles. Users may be given a domain's administrator role. A domain administrator may create tenants, users, and groups within a domain and assign roles to users and groups.

Certificates for PKI

PKI stands for Public Key Infrastructure. Tokens are documents, cryptographically signed using the X509 standard. In order to work correctly token generation requires a public/private key pair. The public key must be signed in an X509 certificate, and the certificate used to sign it must be available as Certificate Authority (CA) certificate. These files can be generated either using the **keystone-manage** utility, or externally generated. The files need to be in the locations specified by the top level Identity Service configuration file `keystone.conf` as specified in the above section. Additionally, the private key should only be readable by the system user that will run the Identity Service.

Warning

The certificates can be world readable, but the private key cannot be. The private key should only be readable by the account that is going to sign tokens. When generating files with the **keystone-manage pki_setup** command, your best option is to run as the pki user. If you run nova-manage as root, you can append –keystone-user and –keystone-group parameters to set the username and group keystone is going to run under.

The values that specify where to read the certificates are under the `[signing]` section of the configuration file. The configuration values are:

- `token_format` - Determines the algorithm used to generate tokens. Can be either `UUID` or `PKI`. Defaults to `PKI`.
- `certfile` - Location of certificate used to verify tokens. Default is `/etc/keystone/ssl/certs/signing_cert.pem`.
- `keyfile` - Location of private key used to sign tokens. Default is `/etc/keystone/ssl/private/signing_key.pem`.
- `ca_certs` - Location of certificate for the authority that issued the above certificate. Default is `/etc/keystone/ssl/certs/ca.pem`.
- `key_size` - Default is `1024`.
- `valid_days` - Default is `3650`.
- `ca_password` - Password required to read the `ca_file`. Default is `None`.

If `token_format=UUID`, a typical token looks like `53f7f6ef0cc344b5be706bcc8b1479e1`. If `token_format=PKI`, a typical token is a much longer string, such as:

```
MIIKtgYJKoZIhvcNAQcCoIIKpzCCCqMCAQExCTAHBgUrDgMCGjCCCY8GCSqGSIb3DQEHAaCCCYAEggl8eyJhY2Nlc3Mi
MFQxNTo1MjowNi43MzMxOTgiLCAiZXhwaXJlcyI6ICIyMDEzLTA1LTMxVDE1OjUyOjA2WiIsICJpZCI6ICJwbGFjZWhv
bCwgImVuYWJsZWQiOiB0cnVlLCAiaWQiOiAiYzJjNTliNGQzZDI4NGQ4ZmEwOWYxNjljYjE4MDBlMDYiLCAibmFtZSI6
b2ludHMiOiBbeyJhZG1pblVSTCI6ICJodHRwOi8vMTkyLjE2OC4yNy4xMDA6ODc3NC92Mi9jMmM1OWI0ZDNkMjg0ZDhm
T25lIiwgImludGVybmFsVVJMIjogImh0dHA6Ly8xOTIuMTY4LjI3LjEwMDo4Nzc0L3YyL2MyYzU5YjRkM2QyODRkOGZh
ODRhNGNhZTk3MmViNzcwOTgzZTJlIiwgInB1YmxpY1VSTCI6ICJodHRwOi8vMTkyLjE2OC4yNy4xMDA6ODc3NC92Mi9j
ImVuZHBvaW50c19saW5rcyI6IFtdLCAidHlwZSI6ICJjb21wdXRlIiwgIm5hbWUiOiAibm92YSJ9LCB7ImVuZHBvaW50
LjEwMDozMzMzIiwgInJlZ2lvbiI6ICJSZWdpb25PbmUiLCAiaW50ZXJuYWxVUkwiOiAiaHR0cDovLzE5Mi4xNjguMjcu
MGU2YWNlNDU4NjZmMzAiLCAicHVibGljVVJMIjogImh0dHA6Ly8xOTIuMTY4LjI3LjEwMDozMzMzIn1dLCAiZW5kcG9p
OiAiczMifSwgeyJlbmRwb2ludHMiOiBbeyJhZG1pblVSTCI6ICJodHRwOi8vMTkyLjE2OC4yNy4xMDA6OTI5MiIsICJy
Imh0dHA6Ly8xOTIuMTY4LjI3LjEwMDo5MjkyIiwgImlkIjogIjczODQzNTJhNTQ0MjQ1NzVhM2NkOTVkN2E0YzNjZGY1
MDA6OTI5MiJ9XSwgImVuZHBvaW50c19saW5rcyI6IFtdLCAidHlwZSI6ICJpbWFnZSIsICJuYW1lIjogImdsYW5jZSJ9
Ly8xOTIuMTY4LjI3LjEwMDo4Nzc2L3YxL2MyYzU5YjRkM2QyODRkOGZhMDlmMTY5Y2IxODAwZTA2IiwgInJlZ2lvbiI6
LzE5Mi4xNjguMjcuMTAwOjg3NzYvdjEvYzJjNTliNGQzZDI4NGQ4ZmEwOWYxNjljYjE4MDBlMDYiLCAiaWQiOiAiMzQ3
bGljVVJMIjogImh0dHA6Ly8xOTIuMTY4LjI3LjEwMDo4Nzc2L3YxL2MyYzU5YjRkM2QyODRkOGZhMDlmMTY5Y2IxODAw
IjogInZvbHVtZSIsICJuYW1lIjogImNpbmRlciJ9LCB7ImVuZHBvaW50cyI6IFt7ImFkbWluVVJMIjogImh0dHA6Ly8x
InJlZ2lvbiI6ICJSZWdpb25PbmUiLCAiaW50ZXJuYWxVUkwiOiAiaHR0cDovLzE5Mi4xNjguMjcuMTAwOjg3NzMvc2Vy
YWE1NDAzMDMzNzI5YzY3MjIiLCAicHVibGljVVJMIjogImh0dHA6Ly8xOTIuMTY4LjI3LjEwMDo4NzczL3NlcnZpY2Vz
eXBlIjogImVjMiIsICJuYW1lIjogImVjMiJ9LCB7ImVuZHBvaW50cyI6IFt7ImFkbWluVVJMIjogImh0dHA6Ly8xOTIu
ZWdpb25PbmUiLCAiaW50ZXJuYWxVUkwiOiAiaHR0cDovLzE5Mi4xNjguMjcuMTAwOjUwMDAvdjIuMCIsICJpZCI6ICJi
dWJsaWNVUkwiOiAiaHR0cDovLzE5Mi4xNjguMjcuMTAwOjUwMDAvdjIuMCJ9XSwgImVuZHBvaW50c19saW5rcyI6IFtd
b25lIn1dLCAidXNlciI6IHsidXNlcm5hbWUiOiAiZGVtbyIsICJyb2xlc19saW5rcyI6IFtdLCAiaWQiOiAiZTVhMTM3
OiBbeyJuYW1lIjogImFub3RoZXJyb2xlIn0sIHsibmFtZSI6ICJNZW1iZXIifV0sICJuYW1lIjogImRlbW8ifSwgIm1l
YWRiODM3NDVkYzQzNGJhMzk5ODllNjBjOTIzYWZhMjgiLCAiMzM2ZTFiNjE1N2Y3NGFmZGJhNWUwYTYwMWUwNjM5MmYi
zCB-AIBATBcMFcxCzAJBgNVBAYTAlVTMQ4wDAYD
VQQIEwVVbnNldDEOMAwGA1UEBxMFVW5zZXQxDjAMBgNVBAoTBVVuc2V0MRgwFgYDVQQDEw93d3cuZXhhbXBsZS5jb20C
nouriuiCgFayIqCssK3SVdhOMINiuJtqv0sE-wBDFiEj-Prcudqlz-n+6q7VgV4mwMPszz39-rwp
+P5l4AjrJasUm7FrO-4l02tPLaaZXU1gBQ1jUG5e5aL5jPDP08HbCWuX6wr-QQQB
SrWY8lF3HrTcJT23sZIleg==
```

Sign certificate issued by external CA

You can use a signing certificate issued by an external CA instead of generated by **keystone-manage**. However, a certificate issued by an external CA must satisfy the following conditions:

- all certificate and key files must be in Privacy Enhanced Mail (PEM) format
- private key files must not be protected by a password

When using signing certificate issued by an external CA, you do not need to specify `key_size`, `valid_days`, and `ca_password` as they will be ignored.

The basic workflow for using a signing certificate issued by an external CA involves:

1. Request Signing Certificate from External CA
2. Convert certificate and private key to PEM if needed
3. Install External Signing Certificate

Request a signing certificate from an external CA

One way to request a signing certificate from an external CA is to first generate a PKCS #10 Certificate Request Syntax (CRS) using OpenSSL CLI.

Create a certificate request configuration file. For example, create the cert_req.conf file, as follows:

```
[ req ]
default_bits            = 1024
default_keyfile         = keystonekey.pem
default_md              = sha1

prompt                  = no
distinguished_name      = distinguished_name

[ distinguished_name ]
countryName             = US
stateOrProvinceName     = CA
localityName            = Sunnyvale
organizationName        = OpenStack
organizationalUnitName  = Keystone
commonName              = Keystone Signing
emailAddress            = keystone@openstack.org
```

Then generate a CRS with OpenSSL CLI. **Do not encrypt the generated private key. Must use the -nodes option.**

For example:

```
$ openssl req -newkey rsa:1024 -keyout signing_key.pem -keyform PEM \
  -out signing_cert_req.pem -outform PEM -config cert_req.conf -nodes
```

If everything is successful, you should end up with signing_cert_req.pem and signing_key.pem. Send signing_cert_req.pem to your CA to request a token signing certificate and make sure to ask the certificate to be in PEM format. Also, make sure your trusted CA certificate chain is also in PEM format.

Install an external signing certificate

Assuming you have the following already:

- signing_cert.pem - (Keystone token) signing certificate in PEM format
- signing_key.pem - corresponding (non-encrypted) private key in PEM format
- cacert.pem - trust CA certificate chain in PEM format

Copy the above to your certificate directory. For example:

```
# mkdir -p /etc/keystone/ssl/certs
# cp signing_cert.pem /etc/keystone/ssl/certs/
# cp signing_key.pem /etc/keystone/ssl/certs/
# cp cacert.pem /etc/keystone/ssl/certs/
# chmod -R 700 /etc/keystone/ssl/certs
```

Note

Make sure the certificate directory is only accessible by root.

If your certificate directory path is different from the default /etc/keystone/ssl/certs, make sure it is reflected in the [signing] section of the configuration file.

Configure the Identity Service with SSL

You can configure the Identity Service to support two-way SSL.

You must obtain the x509 certificates externally and configure them.

The Identity Service provides a set of sample certificates in the `examples/pki/certs` and `examples/pki/private` directories:

Certificate types

cacert.pem	Certificate Authority chain to validate against.
ssl_cert.pem	Public certificate for Identity Service server.
middleware.pem	Public and private certificate for Identity Service middleware/client.
cakey.pem	Private key for the CA.
ssl_key.pem	Private key for the Identity Service server.

Note

You can choose names for these certificates. You can also combine the public/private keys in the same file, if you wish. These certificates are provided as an example.

SSL configuration

To enable SSL with client authentication, modify the `[ssl]` section in the `etc/keystone.conf` file. The following SSL configuration example uses the included sample certificates:

```
[ssl]
enable = True
certfile = <path to keystone.pem>
keyfile = <path to keystonekey.pem>
ca_certs = <path to ca.pem>
cert_required = True
```

Options

- `enable`. True enables SSL. Default is False.
- `certfile`. Path to the Identity Service public certificate file.
- `keyfile`. Path to the Identity Service private certificate file. If you include the private key in the certfile, you can omit the keyfile.
- `ca_certs`. Path to the CA trust chain.
- `cert_required`. Requires client certificate. Default is False.

External authentication with the Identity Service

When the Identity Service runs in `apache-httpd`, you can use external authentication methods that differ from the authentication provided by the identity store back-end. For example, you can use an SQL identity back-end together with X.509 authentication, Kerberos, and so on instead of using the user name and password combination.

Use HTTPD authentication

Web servers, like Apache HTTP, support many methods of authentication. The Identity Service can allow the web server to perform the authentication. The web server then passes the authenticated user to the Identity Service by using the REMOTE_USER environment variable. This user must already exist in the Identity Service back-end so as to get a token from the controller. To use this method, the Identity Service should run on `apache-httpd`.

Use X.509

The following Apache configuration snippet authenticates the user based on a valid X.509 certificate from a known CA:

```
    <VirtualHost _default_:5000>
        SSLEngine on
        SSLCertificateFile    /etc/ssl/certs/ssl.cert
        SSLCertificateKeyFile /etc/ssl/private/ssl.key

        SSLCACertificatePath /etc/ssl/allowed_cas
        SSLCARevocationPath  /etc/ssl/allowed_cas
        SSLUserName          SSL_CLIENT_S_DN_CN
        SSLVerifyClient      require
        SSLVerifyDepth       10

        (...)
    </VirtualHost>
```

Integrate Identity with LDAP

Identity Service supports integration with an existing LDAP directory for authentication and authorization services.

Important

For OpenStack Identity to access an LDAP back end, you must enable the `authlogin_nsswitch_use_ldap` boolean value for SELinux on the Identity server. To enable and make the option persistent across reboots:

```
# setsebool -P authlogin_nsswitch_use_ldap
```

Note

You can integrate Identity with a single LDAP server.

To configure Identity, set options in the `/etc/keystone/keystone.conf` file. Modify these examples as needed.

Procedure 2.1. To integrate Identity with LDAP

1. Enable the LDAP driver in the `keystone.conf` file:

   ```
   [identity]
   #driver = keystone.identity.backends.sql.Identity
   driver = keystone.identity.backends.ldap.Identity
   ```

2. Define the destination LDAP server in the `keystone.conf` file:

   ```
   [ldap]
   url = ldap://localhost
   user = dc=Manager,dc=example,dc=org
   password = samplepassword
   suffix = dc=example,dc=org
   use_dumb_member = False
   allow_subtree_delete = False
   ```

3. Create the organizational units (OU) in the LDAP directory, and define their corresponding location in the `keystone.conf` file:

   ```
   [ldap]
   user_tree_dn = ou=Users,dc=example,dc=org
   user_objectclass = inetOrgPerson

   tenant_tree_dn = ou=Groups,dc=example,dc=org
   tenant_objectclass = groupOfNames

   role_tree_dn = ou=Roles,dc=example,dc=org
   role_objectclass = organizationalRole
   ```

   #### Note

   These schema attributes are extensible for compatibility with various schemas. For example, this entry maps to the `person` attribute in Active Directory:

   ```
   user_objectclass = person
   ```

4. A read-only implementation is recommended for LDAP integration. These permissions are applied to object types in the `keystone.conf` file:

   ```
   [ldap]
   user_allow_create = False
   user_allow_update = False
   user_allow_delete = False

   tenant_allow_create = False
   tenant_allow_update = False
   tenant_allow_delete = False

   role_allow_create = False
   role_allow_update = False
   role_allow_delete = False
   ```

5. Restart the Identity service:

```
# service keystone restart
```

Warning

During service restart, authentication and authorization are unavailable.

Additional LDAP integration settings. Set these options in the `keystone.conf` file.

Filters

Use filters to control the scope of data presented through LDAP.

```
[ldap]
user_filter = (memberof=cn=openstack-users,ou=
workgroups,dc=example,dc=org)
tenant_filter =
role_filter =
```

LDAP Account Status

Mask account status values for compatibility with various directory services. Superfluous accounts are filtered with `user_filter`.

For example, you can mask Active Directory account status attributes in the `keystone.conf` file:

```
[ldap]
user_enabled_attribute = userAccountControl
user_enabled_mask      = 2
user_enabled_default   = 512
```

Separate role authorization and user authentication

When you configure the Identity service to use an LDAP back end, you can split authentication and authorization using the *Assignments* feature.

The Assignments feature enables administrators to manage project role authorization using the Identity service's own SQL database, while still providing user authentication through the LDAP directory.

To configure this:

Procedure 2.2. Separating role authorization and user authentication through Assignments

1. Configure the Identity service to authenticate users through the LDAP driver. To do so, first find the `[identity]` section in the `/etc/keystone/keystone.conf` configuration file. Then, set the `driver` configuration key in that section to `keystone.identity.backends.ldap.Identity`:

```
[identity]
driver = keystone.identity.backends.ldap.Identity
```

2. Next, enable the Assignment driver. To do so, find the `[assignment]` section in the `/etc/keystone/keystone.conf` configuration file. Then, set the `driver` configuration key in that section to `keystone.assignment.backends.sql.Assignment`:

```
[assignment]
driver = keystone.assignment.backends.sql.Assignment
```

On distributions that include openstack-config, you can configure both drivers by running the following commands instead:

```
# openstack-config --set /etc/keystone/keystone.conf \
identity driver keystone.identity.backends.ldap.Identity
# openstack-config --set /etc/keystone/keystone.conf \
assignment driver keystone.assignment.backends.sql.Assignment
```

Secure the OpenStack Identity service connection to an LDAP back end

The Identity service supports the use of TLS to encrypt LDAP traffic. Before configuring this, you must first verify where your certificate authority file is located. For more information, see the section called "Certificates for PKI" [26].

Once you verify the location of your certificate authority file:

Procedure 2.3. Configuring TLS encryption on LDAP traffic

1. Open the `/etc/keystone/keystone.conf` configuration file.

2. Find the `[ldap]` section.

3. In the `[ldap]` section, set the `use_tls` configuration key to `True`. Doing so will enable TLS.

4. Configure the Identity service to use your certificate authorities file. To do so, set the `tls_cacertfile` configuration key in the `ldap` section to the certificate authorities file's path.

   Note

   You can also set the `tls_cacertdir` (also in the `ldap` section) to the directory where all certificate authorities files are kept. If both `tls_cacertfile` and `tls_cacertdir` are set, then the latter will be ignored.

5. Specify what client certificate checks to perform on incoming TLS sessions from the LDAP server. To do so, set the `tls_req_cert` configuration key in the `[ldap]` section to `demand`, `allow`, or `never`:

   - *`demand`*: a certificate will always be requested from the LDAP server. The session will be terminated if no certificate is provided, or if the certificate provided cannot be verified against the existing certificate authorities file.

   - *`allow`*: a certificate will always be requested from the LDAP server. The session will proceed as normal even if a certificate is not provided. If a certificate is provided but it cannot be verified against the existing certificate authorities file, the certificate will be ignored and the session will proceed as normal.

- *never*: a certificate will never be requested.

On distributions that include openstack-config, you can configure TLS encryption on LDAP traffic by running the following commands instead:

```
# openstack --config --set /etc/keystone/keystone.conf \
ldap use_tls True
# openstack-config --set /etc/keystone/keystone.conf \
ldap tls_cacertfile CA_FILE
# openstack-config --set /etc/keystone/keystone.conf \
ldap tls_req_cert CERT_BEHAVIOR
```

Where:

- *CA_FILE* is the absolute path to the certificate authorities file that should be used to encrypt LDAP traffic.
- *CERT_BEHAVIOR*: specifies what client certificate checks to perform on an incoming TLS session from the LDAP server (`demand`, `allow`, or `never`).

Configure Identity service for token binding

Token binding embeds information from an external authentication mechanism, such as a Kerberos server, inside a token. By using token binding, a client can enforce the use of a specified external authentication mechanism with the token. This additional security mechanism ensures that if a token is stolen, for example, it is not usable without external authentication.

You configure the authentication types for a token binding in the `keystone.conf` file:

```
[token]
bind = kerberos
```

Currently only `kerberos` is supported.

To enforce checking of token binding, set the `enforce_token_bind` option to one of these modes:

- `disabled`

  Disables token bind checking.

- `permissive`

  Enables bind checking. If a token is bound to an unknown authentication mechanism, the server ignores it. The default is this mode.

- `strict`

  Enables bind checking. If a token is bound to an unknown authentication mechanism, the server rejects it.

- `required`

Enables bind checking. Requires use of at least authentication mechanism for tokens.

- `named`

  Enables bind checking. Requires use of the specified authentication mechanism for tokens:

```
[token]
enforce_token_bind = kerberos
```

Note

Do not set `enforce_token_bind = named`. The `named` authentication mechanism does not exist.

User CRUD

The Identity Service provides a user CRUD filter that can be added to the public_api pipeline. This user CRUD filter enables users to use a HTTP PATCH to change their own password. To enable this extension you should define a `user_crud_extension` filter, insert it after the `*_body` middleware and before the `public_service` application in the public_api WSGI pipeline in `keystone.conf`. For example:

```
[filter:user_crud_extension]
paste.filter_factory = keystone.contrib.user_crud:CrudExtension.factory

[pipeline:public_api]
pipeline = stats_monitoring url_normalize token_auth admin_token_auth xml_body json_body
 debug ec2_extension user_crud_extension public_service
```

Each user can then change their own password with a HTTP PATCH:

```
$ curl -X PATCH http://localhost:5000/v2.0/OS-KSCRUD/users/<userid> -H
 "Content-type: application/json"  \
  -H "X_Auth_Token: <authtokenid>" -d '{"user": {"password": "ABCD",
 "original_password": "DCBA"}}'
```

In addition to changing their password, all of the user's current tokens are deleted (if the back-end is KVS or sql).

Note

Only use a KVS backend for tokens when testing.

Logging

You configure logging externally to the rest of the Identity Service. The file specifying the logging configuration is in the `[DEFAULT]` section of the `keystone.conf` file under `log_config`. To route logging through syslog, set `use_syslog=true` option in the `[DEFAULT]` section.

A sample logging file is available with the project in the `etc/logging.conf.sample` directory. Like other OpenStack projects, the Identity Service uses the Python logging module, which includes extensive configuration options that let you define the output levels and formats.

Review the `etc/keystone.conf` sample configuration files that are distributed with the Identity Service. For example, each server application has its own configuration file.

For services that have separate paste-deploy `.ini` files, you can configure `auth_token` middleware in the `[keystone_authtoken]` section in the main configuration file, such as `nova.conf`. For example in Compute, you can remove the middleware parameters from `api-paste.ini`, as follows:

```
[filter:authtoken]
paste.filter_factory =
keystoneclient.middleware.auth_token:filter_factory
```

Set these values in the `nova.conf` file:

```
[DEFAULT]
...
auth_strategy=keystone

[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_uri = http://127.0.0.1:5000/
admin_user = admin
admin_password = SuperSekretPassword
admin_tenant_name = service
```

Note

Middleware parameters in paste config take priority. You must remove them to use values in the `[keystone_authtoken]` section.

Monitoring

The Identity Service provides some basic request/response monitoring statistics out of the box.

Enable data collection by defining a `stats_monitoring` filter and including it at the beginning of any desired WSGI pipelines:

```
[filter:stats_monitoring]
paste.filter_factory = keystone.contrib.stats:StatsMiddleware.factory

[pipeline:public_api]
pipeline = stats_monitoring [...] public_service
```

Enable the reporting of collected data by defining a `stats_reporting` filter and including it near the end of your `admin_api` WSGI pipeline (After `*_body` middleware and before `*_extension` filters is recommended):

```
[filter:stats_reporting]
paste.filter_factory = keystone.contrib.stats:StatsExtension.factory

[pipeline:admin_api]
pipeline = [...] json_body stats_reporting ec2_extension [...] admin_service
```

Query the admin API for statistics using:

```
$ curl -H 'X-Auth-Token: ADMIN' http://localhost:35357/v2.0/OS-STATS/stats
```

Reset collected data using:

```
$ curl -H 'X-Auth-Token: ADMIN' -X DELETE \
          http://localhost:35357/v2.0/OS-STATS/stats
```

Start the Identity Service

To start the services for the Identity Service, run the following command:

```
$ keystone-all
```

This command starts two wsgi.Server instances configured by the `keystone.conf` file as described previously. One of these wsgi servers is `admin` (the administration API) and the other is `main` (the primary/public API interface). Both run in a single process.

Example usage

The `keystone` client is set up to expect commands in the general form of `keystone command argument`, followed by flag-like keyword arguments to provide additional (often optional) information. For example, the command `user-list` and `tenant-create` can be invoked as follows:

```
# Using token auth env variables
export OS_SERVICE_ENDPOINT=http://127.0.0.1:5000/v2.0/
export OS_SERVICE_TOKEN=secrete_token
keystone user-list
keystone tenant-create --name=demo

# Using token auth flags
keystone --os-token=secrete --os-endpoint=http://127.0.0.1:5000/v2.0/ user-list
keystone --os-token=secrete --os-endpoint=http://127.0.0.1:5000/v2.0/ tenant-create --name=demo

# Using user + password + tenant_name env variables
export OS_USERNAME=admin
export OS_PASSWORD=secrete
export OS_TENANT_NAME=admin
keystone user-list
keystone tenant-create --name=demo

# Using user + password + tenant_name flags
keystone --username=admin --password=secrete --tenant_name=admin user-list
keystone --username=admin --password=secrete --tenant_name=admin tenant-create --name=demo
```

Authentication middleware with user name and password

You can also configure the Identity Service authentication middleware using the `admin_user` and `admin_password` options. When using the `admin_user` and `admin_password` options the `admin_token` parameter is optional. If `admin_token` is specified, it is used only if the specified token is still valid.

For services that have a separate paste-deploy .ini file, you can configure the authentication middleware in the `[keystone_authtoken]` section of the main configuration file, such as `nova.conf`. In Compute, for example, you can remove the middleware parameters from `api-paste.ini`, as follows:

```
[filter:authtoken]
paste.filter_factory =
keystoneclient.middleware.auth_token:filter_factory
```

And set the following values in `nova.conf` as follows:

```
[DEFAULT]
...
auth_strategy=keystone

[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
auth_uri = http://127.0.0.1:5000/
admin_user = admin
admin_password = SuperSekretPassword
admin_tenant_name = service
```

Note

The middleware parameters in the paste config take priority. You must remove them to use the values in the [keystone_authtoken] section.

This sample paste config filter makes use of the `admin_user` and `admin_password` options:

```
[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
service_port = 5000
service_host = 127.0.0.1
auth_port = 35357
auth_host = 127.0.0.1
auth_token = 012345SECRET99TOKEN012345
admin_user = admin
admin_password = keystone123
```

Note

Using this option requires an admin tenant/role relationship. The admin user is granted access to the admin role on the admin tenant.

Troubleshoot the Identity Service

To troubleshoot the Identity Service, review the logs in the `/var/log/keystone.log` file.

Note

Use the `/etc/keystone/logging.conf` file to configure the location of log files.

The logs show the components that have come in to the WSGI request, and ideally show an error that explains why an authorization request failed. If you do not see the request in the logs, run keystone with `--`*debug* parameter. Pass the `--`*debug* parameter before the command parameters.

Debug PKI middleware

If you receive an `Invalid OpenStack Identity Credentials` message when you talk to an OpenStack service, it might be caused by the changeover from UUID tokens to PKI tokens in the Grizzly release. Learn how to troubleshoot this error.

The PKI-based token validation scheme relies on certificates from the Identity Service that are fetched through HTTP and stored in a local directory. The location for this directory is specified by the `signing_dir` configuration option. In your services configuration file, look for a section like this:

```
[keystone_authtoken]
signing_dir = /var/cache/glance/api
auth_uri = http://127.0.0.1:5000/
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = glance
```

If your service lacks this stanza, the keystoneclient/middleware/auth_token.py file specifies the defaults. If no value is specified for this directory, it defaults to a secure temporary directory. Initialization code for the service checks that the directory exists and is writable. If it does not exist, the code tries to create it. If this fails, the service fails to start. However, it often succeeds but problems occur later.

The first thing to check is that the `signing_dir` does, in fact, exist. If it does, check for the presence of the certificate files inside there:

```
$ ls -la /var/cache/glance/api/
```

```
total 24
drwx------. 2 ayoung root 4096 Jul 22 10:58 .
drwxr-xr-x. 4 root root 4096 Nov 7 2012 ..
-rw-r-----. 1 ayoung ayoung 1424 Jul 22 10:58 cacert.pem
-rw-r-----. 1 ayoung ayoung 15 Jul 22 10:58 revoked.pem
-rw-r-----. 1 ayoung ayoung 4518 Jul 22 10:58 signing_cert.pem
```

This directory contains two certificates and the token revocation list. If these files are not present, your service cannot fetch them from the Identity Service. To troubleshoot, try to talk to the Identity Service to make sure it is serving out the files, as follows:

```
$ curl http://localhost:35357/v2.0/certificates/signing
```

This command fetches the signing certificate:

```
Certificate:
    Data:
        Version: 3 (0x2)
        Serial Number: 1 (0x1)
    Signature Algorithm: sha1WithRSAEncryption
        Issuer: C=US, ST=Unset, L=Unset, O=Unset, CN=www.example.com
        Validity
            Not Before: Jul 22 14:57:31 2013 GMT
            Not After : Jul 20 14:57:31 2023 GMT
        Subject: C=US, ST=Unset, O=Unset, CN=www.example.com
```

Note the expiration dates of the certificate:

```
Not Before: Jul 22 14:57:31 2013 GMT
Not After : Jul 20 14:57:31 2023 GMT
```

The token revocation list is updated once a minute, but the certificates are not. One possible problem is that the certificates are the wrong files or garbage. You can remove these files and run another command against your server: They are fetched on demand.

The Identity Service log should show the access of the certificate files. You might have to turn up your logging levels. Set `debug = True` and `verbose = True` in your Identity Service configuration file and restart the Identity Service server.

```
(keystone.common.wsgi): 2013-07-24 12:18:11,461 DEBUG wsgi __call__
arg_dict: {}
(access): 2013-07-24 12:18:11,462 INFO core __call__ 127.0.0.1 - - [24/Jul/
2013:16:18:11 +0000]
"GET http://localhost:35357/v2.0/certificates/signing HTTP/1.0" 200 4518
```

If the files do not appear in your directory after this, it is likely one of the following issues:

- Your service is configured incorrectly and cannot talk to the Identity Service. Check the `auth_port` and `auth_host` values and make sure that you can talk to that service through cURL, as shown previously.
- Your signing directory is not writable. Use the **chmod** command to change its permissions so that the service (POSIX) user can write to it. Verify the change through **su** and **touch** commands.
- The SELinux policy is denying access to the directory.

SELinux troubles often occur when you use Fedora/RHEL-based packages and you choose configuration options that do not match the standard policy. Run the **setenforce permissive** command. If that makes a difference, you should relabel the directory. If you are using a sub-directory of the `/var/cache/` directory, run the following command:

```
# restorecon /var/cache/
```

If you are not using a `/var/cache` sub-directory, you should. Modify the `signing_dir` configuration option for your service and restart.

Set back to `setenforce enforcing` to confirm that your changes solve the problem.

If your certificates are fetched on demand, the PKI validation is working properly. Most likely, the token from the Identity Service is not valid for the operation you are attempting to perform, and your user needs a different role for the operation.

Debug signing key file errors

If an error occurs when the signing key file opens, it is possible that the person who ran the **keystone-manage pki_setup** command to generate certificates and keys did not use the correct user. When you run the **keystone-manage pki_setup** command, the Identity Service generates a set of certificates and keys in `/etc/keystone/ssl*`, which is owned by root:root.

This can present a problem when you run the Identity Service daemon under the keystone user account (nologin) when you try to run PKI. Unless you run the **chown** command against the files keystone:keystone or run the **keystone-manage pki_setup** command with the `--keystone-user` and `--keystone-group` parameters, you get an error, as follows:

```
2012-07-31 11:10:53 ERROR [keystone.common.cms] Error opening signing key file
/etc/keystone/ssl/private/signing_key.pem
140380567730016:error:0200100D:system library:fopen:Permission
denied:bss_file.c:398:fopen('/etc/keystone/ssl/private/signing_key.pem','r')
140380567730016:error:20074002:BIO routines:FILE_CTRL:system lib:bss_file.c:400:
unable to load signing key file
```

3. Dashboard

Table of Contents

The OpenStack dashboard is a web-based interface that allows you to manage OpenStack resources and services. The dashboard allows you to interact with the OpenStack Compute cloud controller using the OpenStack APIs. For more information about installing and configuring the dashboard, see the *OpenStack Installation Guide* for your operating system.

For more information about using the dashboard, see:

- the section called "Customize the dashboard" [42], for customizing the dashboard.
- the section called "Set up session storage for the dashboard" [44], for setting up session storage for the dashboard.
- The Horizon documentation, for deploying the dashboard.
- The *OpenStack End User Guide*, for launching instances with the dashboard..

Customize the dashboard

Adapted from How To Custom Brand The OpenStack "Horizon" Dashboard.

You install the OpenStack dashboard through the `openstack-dashboard` package. You can customize the dashboard with your own colors, logo, and site title through a CSS file.

Canonical also provides an `openstack-dashboard-ubuntu-theme` package that brands the Python-based Django interface.

1. Create a graphical logo with a transparent background. The text `TGen Cloud` in this example is rendered through `.png` files of multiple sizes created with a graphics program.

   Use a 200×27 for the logged-in banner graphic, and 365×50 for the login screen graphic.

2. Set the HTML title, which appears at the top of the browser window, by adding the following line to `/etc/openstack-dashboard/local_settings.py`:

   ```
   SITE_BRANDING = "Example, Inc. Cloud"
   ```

3. Upload your new graphic files to the following location: `/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/img/`

4. Create a CSS style sheet in the following directory: `/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/css/`

5. Edit your CSS file to override the Ubuntu customizations in the `ubuntu.css` file.

   Change the colors and image file names as appropriate, though the relative directory paths should be the same. The following example file shows you how to customize your CSS file:

```
/*
* New theme colors for dashboard that override the defaults:
*  dark blue: #355796 / rgb(53, 87, 150)
*  light blue: #BAD3E1 / rgb(186, 211, 225)
*
* By Preston Lee <plee@tgen.org>
*/
h1.brand {
background: #355796 repeat-x top left;
border-bottom: 2px solid #BAD3E1;
}
h1.brand a {
background: url(../img/my_cloud_logo_small.png) top left no-repeat;
}
#splash .login {
background: #355796 url(../img/my_cloud_logo_medium.png) no-repeat center 35px;
}
#splash .login .modal-header {
border-top: 1px solid #BAD3E1;
}
.btn-primary {
background-image: none !important;
background-color: #355796 !important;
border: none !important;
box-shadow: none;
}
.btn-primary:hover,
.btn-primary:active {
border: none;
box-shadow: none;
background-color: #BAD3E1 !important;
text-decoration: none;
}
```

6. Open the following HTML template in an editor: `/usr/share/openstack-dashboard/openstack_dashboard/templates/_stylesheets.html`

7. Add a line to include your `custom.css` file:

```
...
 <link href='{{ STATIC_URL }}bootstrap/css/bootstrap.min.css' media='screen' rel='stylesheet' />
 <link href='{{ STATIC_URL }}dashboard/css/{% choose_css %}' media='screen' rel='stylesheet' />
 <link href='{{ STATIC_URL }}dashboard/css/custom.css' media='screen' rel='stylesheet' />
 ...
```

8. Restart apache:

   On Ubuntu:

```
# service apache2 restart
```

   On Fedora, RHEL, CentOS:

```
# service httpd restart
```

   On openSUSE:

```
# service apache2 restart
```

9. Reload the dashboard in your browser to view your changes.

   Modify your CSS file as appropriate.

Set up session storage for the dashboard

The dashboard uses Django sessions framework to handle user session data. However, you can use any available session back end. You customize the session back end through the `SESSION_ENGINE` setting in your `local_settings` file (on Fedora/RHEL/CentOS: `/etc/openstack-dashboard/local_settings`, on Ubuntu and Debian: `/etc/openstack-dashboard/local_settings.py` and on openSUSE: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`).

The following sections describe the pros and cons of each option as it pertains to deploying the dashboard.

Local memory cache

Local memory storage is the quickest and easiest session back end to set up, as it has no external dependencies whatsoever. It has the following significant drawbacks:

- No shared storage across processes or workers.
- No persistence after a process terminates.

The local memory back end is enabled as the default for Horizon solely because it has no dependencies. It is not recommended for production use, or even for serious development work. Enabled by:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'BACKEND': 'django.core.cache.backends.locmem.LocMemCache'
}
```

Key-value stores

You can use applications such as Memcached or Redis for external caching. These applications offer persistence and shared storage and are useful for small-scale deployments and/or development.

Memcached

Memcached is a high-performance and distributed memory object caching system providing in-memory key-value store for small chunks of arbitrary data.

Requirements:

- Memcached service running and accessible.
- Python module `python-memcached` installed.

Enabled by:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    'BACKEND': 'django.core.cache.backends.memcached.MemcachedCache'
    'LOCATION': 'my_memcached_host:11211',
}
```

Redis

Redis is an open source, BSD licensed, advanced key-value store. It is often referred to as a data structure server.

Requirements:

- Redis service running and accessible.
- Python modules `redis` and `django-redis` installed.

Enabled by:

```
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'
CACHES = {
    "default": {
        "BACKEND": "redis_cache.cache.RedisCache",
        "LOCATION": "127.0.0.1:6379:1",
        "OPTIONS": {
            "CLIENT_CLASS": "redis_cache.client.DefaultClient",
        }
    }
}
```

Initialize and configure the database

Database-backed sessions are scalable, persistent, and can be made high-concurrency and highly-available.

However, database-backed sessions are one of the slower session storages and incur a high overhead under heavy usage. Proper configuration of your database deployment can also be a substantial undertaking and is far beyond the scope of this documentation.

1. Start the mysql command-line client:

   ```
   $ mysql -u root -p
   ```

2. Enter the MySQL root user's password when prompted.

3. To configure the MySQL database, create the dash database:

   ```
   mysql> CREATE DATABASE dash;
   ```

4. Create a MySQL user for the newly-created dash database that has full control of the database. Replace *DASH_DBPASS* with a password for the new user:

   ```
   mysql> GRANT ALL PRIVILEGES ON dash.* TO 'dash'@'%' IDENTIFIED BY
    'DASH_DBPASS';
   mysql> GRANT ALL PRIVILEGES ON dash.* TO 'dash'@'localhost' IDENTIFIED BY
    'DASH_DBPASS';
   ```

5. Enter quit at the `mysql>` prompt to exit MySQL.

6. In the `local_settings` file (on Fedora/RHEL/CentOS: `/etc/openstack-dashboard/local_settings`, on Ubuntu/Debian: `/etc/openstack-dashboard/local_settings.py` and on openSUSE: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`), change these options:

```
SESSION_ENGINE = 'django.core.cache.backends.db.DatabaseCache'
DATABASES = {
    'default': {
        # Database configuration here
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'DASH_DBPASS',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    }
}
```

7. After configuring the `local_settings` as shown, you can run the **manage.py syncdb** command to populate this newly-created database.

   ```
   $ /usr/share/openstack-dashboard/manage.py syncdb
   ```

   Note on openSUSE the path is `/srv/www/openstack-dashboard/manage.py`.

   As a result, the following output is returned:

   ```
   Installing custom SQL ...
   Installing indexes ...
   DEBUG:django.db.backends:(0.008) CREATE INDEX `django_session_c25c2c28` ON
    `django_session` (`expire_date`);; args=()
   No fixtures found.
   ```

8. On Ubuntu: If you want to avoid a warning when you restart apache2, create a blackhole directory in the dashboard directory, as follows:

   ```
   # mkdir -p /var/lib/dash/.blackhole
   ```

9. Restart Apache to pick up the default site and symbolic link settings:

   On Ubuntu:

   ```
   # /etc/init.d/apache2 restart
   ```

   On Fedora/RHEL/CentOS:

   ```
   # service httpd restart
   ```

   ```
   # service apache2 restart
   ```

   On openSUSE:

   ```
   # systemctl restart apache2.service
   ```

10. On Ubuntu, restart the `nova-api` service to ensure that the API server can connect to the dashboard without error:

    ```
    # service nova-api restart
    ```

Cached database

To mitigate the performance issues of database queries, you can use the Django **cached_db** session back end, which utilizes both your database and caching infrastructure to perform write-through caching and efficient retrieval.

Enable this hybrid setting by configuring both your database and cache, as discussed previously. Then, set the following value:

```
SESSION_ENGINE = "django.contrib.sessions.backends.cached_db"
```

Cookies

If you use Django 1.4 or later, the **signed_cookies** back end avoids server load and scaling problems.

This back end stores session data in a cookie, which is stored by the user's browser. The back end uses a cryptographic signing technique to ensure session data is not tampered with during transport. This is not the same as encryption; session data is still readable by an attacker.

The pros of this engine are that it requires no additional dependencies or infrastructure overhead, and it scales indefinitely as long as the quantity of session data being stored fits into a normal cookie.

The biggest downside is that it places session data into storage on the user's machine and transports it over the wire. It also limits the quantity of session data that can be stored.

See the Django cookie-based sessions documentation.

4. Compute

Table of Contents

The OpenStack Compute service allows you to control an Infrastructure-as-a-Service (IaaS) cloud computing platform. It gives you control over instances and networks, and allows you to manage access to the cloud through users and projects.

Compute does not include virtualization software. Instead, it defines drivers that interact with underlying virtualization mechanisms that run on your host operating system, and exposes functionality over a web-based API.

System architecture

OpenStack Compute contains several main components.

- The *cloud controller* represents the global state and interacts with the other components. The `API server` acts as the web services front end for the cloud controller. The `compute controller` provides compute server resources and usually also contains the Compute service.
- The `object store` is an optional component that provides storage services; you can also instead use OpenStack Object Storage.
- An `auth manager` provides authentication and authorization services when used with the Compute system; you can also instead use OpenStack Identity as a separate authentication service.
- A `volume controller` provides fast and permanent block-level storage for the compute servers.
- The `network controller` provides virtual networks to enable compute servers to interact with each other and with the public network. You can also instead use OpenStack Networking.

- The `scheduler` is used to select the most suitable compute controller to host an instance.

Compute uses a messaging-based, `shared nothing` architecture. All major components exist on multiple servers, including the compute,volume, and network controllers, and the object store or image service. The state of the entire system is stored in a database. The cloud controller communicates with the internal object store using HTTP, but it communicates with the scheduler, network controller, and volume controller using AMQP (advanced message queueing protocol). To avoid blocking a component while waiting for a response, Compute uses asynchronous calls, with a callback that is triggered when a response is received.

Hypervisors

Compute controls hypervisors through an API server. Selecting the best hypervisor to use can be difficult, and you must take budget, resource constraints, supported features, and required technical specifications into account. However, the majority of OpenStack development is done on systems using KVM and Xen-based hypervisors. For a detailed list of features and support across different hypervisors, see http://wiki.openstack.org/HypervisorSupportMatrix.

You can also orchestrate clouds using multiple hypervisors in different availability zones. Compute supports the following hypervisors:

- Baremetal
- Docker
- Hyper-V
- Kernel-based Virtual Machine (KVM)
- Linux Containers (LXC)
- Quick Emulator (QEMU)
- User Mode Linux (UML)
- VMWare vSphere
- Xen

For more information about hypervisors, see the Hypervisors section in the *OpenStack Configuration Reference*.

Tenants, users, and roles

The Compute system is designed to be used by different consumers in the form of tenants on a shared system, and role-based access assignments. Roles control the actions that a user is allowed to perform.

Tenants are isolated resource containers that form the principal organizational structure within the Compute service. They consist of an individual VLAN, and volumes, instances, images, keys, and users. A user can specify the tenant by appending `:project_id` to their access key. If no tenant is specified in the API request, Compute attempts to use a tenant with the same ID as the user.

For tenants, you can use quota controls to limit the:

- Number of volumes that may be launched.
- Number of processor cores and the amount of RAM that can be allocated.
- Floating IP addresses assigned to any instance when it launches. This allows instances to have the same publicly accessible IP addresses.
- Fixed IP addresses assigned to the same instance when it launches. This allows instances to have the same publicly or privately accessible IP addresses.

Roles control the actions a user is allowed to perform. By default, most actions do not require a particular role, but you can configure them by editing the `policy.json` file for user roles. For example, a rule can be defined so that a user must have the *`admin`* role in order to be able to allocate a public IP address.

A tenant limits users' access to particular images. Each user is assigned a username and password. Keypairs granting access to an instance are enabled for each user, but quotas are set, so that each tenant can control resource consumption across available hardware resources.

Note

Earlier versions of OpenStack used the term `project` instead of `tenant`. Because of this legacy terminology, some command-line tools use `--`*`project_id`* where you would normally expect to enter a tenant ID.

Images and instances

Disk images provide templates for virtual machine file systems. The Image Service manages storage and management of images.

Instances are the individual virtual machines that run on physical compute nodes. Users can launch any number of instances from the same image. Each launched instance runs from a copy of the base image so that any changes made to the instance do not affect the base image. You can take snapshots of running instances to create an image based on the current disk state of a particular instance. The Compute services manages instances.

When you launch an instance, you must choose a `flavor`, which represents a set of virtual resources. Flavors define how many virtual CPUs an instance has and the amount of RAM and size of its ephemeral disks. OpenStack provides a number of predefined flavors that you can edit or add to. Users must select from the set of available flavors defined on their cloud.

Note

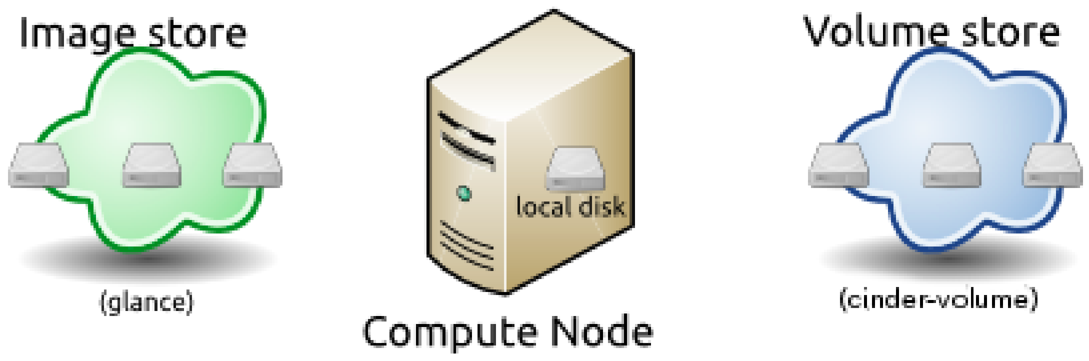
- For more information about creating and troubleshooting images, see the *OpenStack Virtual Machine Image Guide*.

- For more information about image configuration options, see the Image Services section of the *OpenStack Configuration Reference*.
- For more information about flavors, see the section called “Flavors” [58] or the Flavors section in the *OpenStack Operations Guide*.

You can add and remove additional resources from running instances, such as persistent volume storage, or public IP addresses. The example used in this chapter is of a typical virtual system within an OpenStack cloud. It uses the `cinder-volume` service, which provides persistent block storage, instead of the ephemeral storage provided by the selected instance flavor.

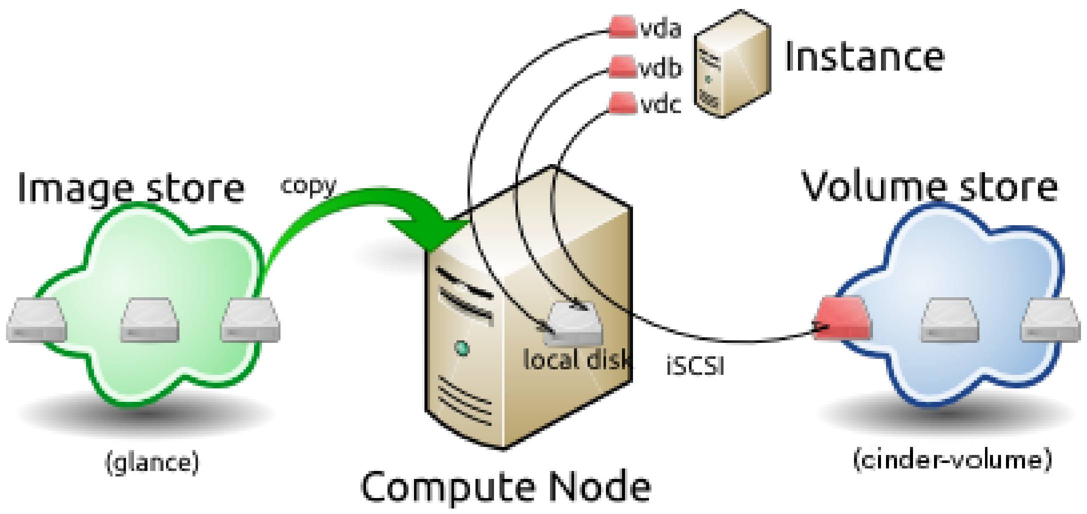
This diagram shows the system state prior to launching an instance. The image store, fronted by the Image service (glance) has a number of predefined images. Inside the cloud, a compute node contains the available vCPU, memory, and local disk resources. Additionally, the `cinder-volume` service provides a number of predefined volumes.

Figure 4.1. Base image state with no running instances



To launch an instance, select an image, a flavor, and other optional attributes. The selected flavor provides a root volume, labeled `vda` in this diagram, and additional ephemeral storage, labeled `vdb`. In this example, the `cinder-volume` store is mapped to the third virtual disk on this instance, `vdc`.

Figure 4.2. Instance creation from image and runtime state



The base image is copied from the image store to the local disk. The local disk is the first disk that the instance accesses, and is labeled `vda`. By using smaller images, your instances start up faster as less data needs to be copied across the network.

A new empty disk, labeled `vdb` is also created. This is an empty ephemeral disk, which is destroyed when you delete the instance.

The compute node is attached to the `cinder-volume` using iSCSI, and maps to the third disk, `vdc`. The vCPU and memory resources are provisioned and the instance is booted from `vda`. The instance runs and changes data on the disks as indicated in red in the diagram.

Note

Some of the details in this example scenario might be different in your environment. For example, you might use a different type of back-end storage or different network protocols. One common variant is that the ephemeral storage used for volumes `vda` and `vdb` could be backed by network storage rather than a local disk.

When the instance is deleted, the state is reclaimed with the exception of the persistent volume. The ephemeral storage is purged; memory and vCPU resources are released. The image remains unchanged throughout.

Figure 4.3. End state of image and volume after instance exits

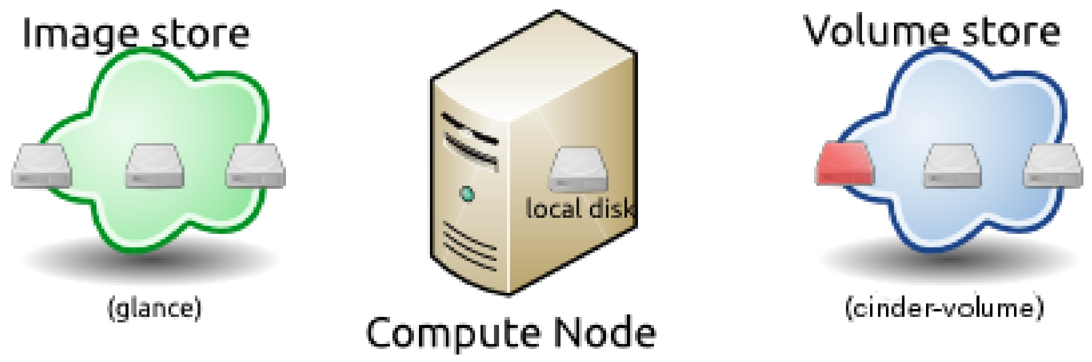


Image management

The OpenStack Image Service discovers, registers, and retrieves virtual machine images. The service also includes a RESTful API that allows you to query VM image metadata and retrieve the actual image with HTTP requests. For more information about the API, see the OpenStack API Complete Reference and the Python API.

The OpenStack Image Service can be controlled using a command-line tool. For more information about the using OpenStack Image command-line tool, see the Manage Images section in the *OpenStack End User Guide*.

Virtual images that have been made available through the Image Service can be stored in a variety of ways. In order to use these services, you must have a working installation of the Image Service, with a working endpoint, and users that have been created in OpenStack Identity. Additionally, you must meet the environment variables required by the Compute and Image Service clients.

The Image Service supports these back-end stores:

File system	The OpenStack Image Service stores virtual machine images in the file system back end by default. This simple back end writes image files to the local file system.
Object Storage service	The OpenStack highly available service for storing objects.
S3	The Amazon S3 service.
HTTP	OpenStack Image Service can read virtual machine images that are available on the internet using HTTP. This store is read only.
Rados block device (RBD)	Stores images inside of a Ceph storage cluster using Ceph's RBD interface.
GridFS	Stores images using MongoDB.

Image property protection

There are currently two types of properties in the Image Service: "core properties," which are defined by the system, and "additional properties," which are arbitrary key/value pairs that can be set on an image.

Any such property can be protected through configuration. When you put protections on a property, it limits the users who can perform CRUD operations on the property based on their user role. The use case is to enable the cloud provider to maintain extra properties on images. Typically this would be performed by an administrator who has access to protected properties, managed in the `policy.json` file. The extra property could be licensing information or billing information, for example.

Properties that don't have protections defined for them will act as they do now: the administrator can control core properties, with the image owner having control over additional properties.

Property protection can be set in `/etc/glance/property-protections.conf`, using roles found in `policy.json`.

Instance building blocks

In OpenStack, the base operating system is usually copied from an image stored in the OpenStack Image Service. This is the most common case and results in an ephemeral instance that starts from a known template state and loses all accumulated states on shutdown.

You can also put an operating system on a persistent volume in Compute or the Block Storage volume system. This gives a more traditional, persistent system that accumulates states, which are preserved across restarts. To get a list of available images on your system, run:

```
$ nova image-list
+--------------------------------------+-------------------------------+--------+--------------------------------------+
| ID                                   | Name                          | Status | Server                               |
+--------------------------------------+-------------------------------+--------+--------------------------------------+
| aee1d242-730f-431f-88c1-87630c0f07ba | Ubuntu 12.04 cloudimg amd64   | ACTIVE |                                      |
| 0b27baa1-0ca6-49a7-b3f4-48388e440245 | Ubuntu 12.10 cloudimg amd64   | ACTIVE |                                      |
| df8d56fc-9cea-4dfd-a8d3-28764de3cb08 | jenkins                       | ACTIVE |                                      |
+--------------------------------------+-------------------------------+--------+--------------------------------------+
```

The displayed image attributes are:

`ID`	Automatically generated UUID of the image.
`Name`	Free form, human-readable name for image.
`Status`	The status of the image. Images marked `ACTIVE` are available for use.
`Server`	For images that are created as snapshots of running instances, this is the UUID of the instance the snapshot derives from. For uploaded images, this field is blank.

Virtual hardware templates are called `flavors`. The default installation provides five flavors. By default, these are configurable by administrative users. However, you can change this behavior by redefining the access controls for *`compute_extension:flavormanage`* in `/etc/nova/policy.json` on the `compute-api` server.

For a list of flavors that are available on your system, run:

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+
| ID |    Name   | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |
+----+-----------+-----------+------+-----------+------+-------+-------------+
| 1  | m1.tiny   | 512       | 1    | N/A       | 0    | 1     |             |
| 2  | m1.small  | 2048      | 20   | N/A       | 0    | 1     |             |
| 3  | m1.medium | 4096      | 40   | N/A       | 0    | 2     |             |
| 4  | m1.large  | 8192      | 80   | N/A       | 0    | 4     |             |
| 5  | m1.xlarge | 16384     | 160  | N/A       | 0    | 8     |             |
+----+-----------+-----------+------+-----------+------+-------+-------------+
```

Instance management tools

OpenStack provides command-line, web-based, and API-based instance management tools. Additionally, a number of third-party management tools are available, using either the native API or the provided EC2-compatible API.

The OpenStack python-novaclient package provides a basic command-line utility, which uses the **nova** command. This is available as a native package for most Linux distributions, or you can install the latest version using the pip python package installer:

```
# pip install python-novaclient
```

For more information about python-novaclient and other available command-line tools, see the *OpenStack End User Guide*.

```
$ nova --debug list
 connect: (10.0.0.15, 5000)
send: 'POST /v2.0/tokens HTTP/1.1\r\nHost: 10.0.0.15:5000\r\nContent-Length: 116\
r\ncontent-type: application/json\r\naccept-encoding: gzip, deflate\r\naccept:
 application/json\r\nuser-agent: python-novaclient\r\n\r\n{"auth": {"tenantName":
 "demoproject", "passwordCredentials": {"username": "demouser", "password":
 "demopassword"}}}'
reply: 'HTTP/1.1 200 OK\r\n'
header: Content-Type: application/json
header: Vary: X-Auth-Token
header: Date: Thu, 13 Sep 2012 20:27:36 GMT
header: Transfer-Encoding: chunked
connect: (128.52.128.15, 8774)
send: u'GET /v2/fa9dccdeadbeef23ae230969587a14bf/servers/detail HTTP/1.1\
r\nHost: 10.0.0.15:8774\r\nx-auth-project-id: demoproject\r\nx-auth-token:
 deadbeef9998823afecc3d552525c34c\r\naccept-encoding: gzip, deflate\r\naccept:
 application/json\r\nuser-agent: python-novaclient\r\n\r\n'
reply: 'HTTP/1.1 200 OK\r\n'
header: X-Compute-Request-Id: req-bf313e7d-771a-4c0b-ad08-c5da8161b30f
header: Content-Type: application/json
header: Content-Length: 15
header: Date: Thu, 13 Sep 2012 20:27:36 GMT
  !!removed matrix for validation!!
```

Control where instances run

The *OpenStack Configuration Reference* provides detailed information on controlling where your instances run, including ensuring a set of instances run on different compute nodes for service resiliency or on the same node for high performance inter-instance communications.

Admin users can specify an exact compute node to run on using the command **--availability-zone** ***`availability-zone`***:***`compute-host`***

Compute service node firewall requirements

Console connections for virtual machines, whether direct or through a proxy, are received on ports `5900` to `5999`. You must configure the firewall on each Compute service node to enable network traffic on these ports.

Procedure 4.1. Configure the service-node firewall

1. On the server that hosts the Compute service, log in as `root`.
2. Edit the `/etc/sysconfig/iptables` file.
3. Add an INPUT rule that allows TCP traffic on ports that range from `5900` to `5999`:

   ```
   -A INPUT -p tcp -m multiport --dports 5900:5999 -j ACCEPT
   ```

   The new rule must appear before any INPUT rules that REJECT traffic.
4. Save the changes to the `/etc/sysconfig/iptables` file.
5. Restart the `iptables` service to ensure that the change takes effect.

   ```
   $ service iptables restart
   ```

The `iptables` firewall now enables incoming connections to the Compute services. Repeat this process for each Compute service node.

Block storage

OpenStack provides two classes of block storage: ephemeral storage and persistent volumes. Volumes are persistent virtualized block devices independent of any particular instance.

Ephemeral storage is associated with a single unique instance, and it exists only for the life of that instance. The amount of ephemeral storage is defined by the flavor of the instance. Generally, the root file system for an instance will be stored on ephemeral storage. It persists across reboots of the guest operating system, but when the instance is deleted, the ephemeral storage is also removed.

In addition to the ephemeral root volume, all flavors except the smallest, `m1.tiny`, also provide an additional ephemeral block device of between 20 and 160 GB. These sizes can be configured to suit your environment. This is presented as a raw block device with no partition table or file system. Cloud-aware operating system images can discover, format, and mount these storage devices. For example, the `cloud-init` package included in Ubuntu's stock cloud images format this space as an `ext3` file system and mount it on `/`

`mnt`. This is a feature of the guest operating system you are using, and is not an OpenStack mechanism. OpenStack only provisions the raw storage.

Persistent volumes are created by users and their size is limited only by the user's quota and availability limits. Upon initial creation, volumes are raw block devices without a partition table or a file system. To partition or format volumes, you must attach them to an instance. Once they are attached to an instance, you can use persistent volumes in much the same way as you would use external hard disk drive. You can attach volumes to only one instance at a time, although you can detach and reattach volumes to as many different instances as you like.

You can configure persistent volumes as bootable and use them to provide a persistent virtual instance similar to traditional non-cloud-based virtualization systems. Typically, the resulting instance can also still have ephemeral storage depending on the flavor selected, but the root file system can be on the persistent volume and its state maintained even if the instance is shut down. For more information about this type of configuration, see the *OpenStack Configuration Reference*.

Note

Persistent volumes do not provide concurrent access from multiple instances. That type of configuration requires a traditional network file system like NFS or CIFS, or a cluster file system such as GlusterFS. These systems can be built within an OpenStack cluster or provisioned outside of it, but OpenStack software does not provide these features.

EC2 compatibility API

In addition to the native compute API, OpenStack provides an EC2-compatible API. This API allows EC2 legacy workflows built for EC2 to work with OpenStack. The *OpenStack Configuration Reference* lists configuration options for customizing this compatibility API on your OpenStack cloud.

Numerous third-party tools and language-specific SDKs can be used to interact with OpenStack clouds, using both native and compatibility APIs. Some of the more popular third-party tools are:

Euca2ools	A popular open source command-line tool for interacting with the EC2 API. This is convenient for multi-cloud environments where EC2 is the common API, or for transitioning from EC2-based clouds to OpenStack. For more information, see the euca2ools site.
Hybridfox	A Firefox browser add-on that provides a graphical interface to many popular public and private cloud technologies, including OpenStack. For more information, see the hybridfox site.
boto	A Python library for interacting with Amazon Web Services. It can be used to access OpenStack through the EC2 compatibility API. For more information, see the boto project page on GitHub.
fog	A Ruby cloud services library. It provides methods for interacting with a large number of cloud and virtualization platforms, including OpenStack. For more information, see the fog site.

php-opencloud	A PHP SDK designed to work with most OpenStack- based cloud deployments, as well as Rackspace public cloud. For more information, see the php-opencloud site.

Building blocks

In OpenStack the base operating system is usually copied from an image stored in the OpenStack Image Service. This is the most common case and results in an ephemeral instance that starts from a known template state and loses all accumulated states on shutdown. It is also possible to put an operating system on a persistent volume in the Nova-Volume or Cinder volume system. This gives a more traditional persistent system that accumulates states, which are preserved across restarts. To get a list of available images on your system run:

```
$ nova image-list
+--------------------------------------+-------------------------------+--------+--------------------------------------+
| ID                                   | Name                          | Status | Server                               |
+--------------------------------------+-------------------------------+--------+--------------------------------------+
| aee1d242-730f-431f-88c1-87630c0f07ba | Ubuntu 12.04 cloudimg amd64   | ACTIVE |                                      |
| 0b27baa1-0ca6-49a7-b3f4-48388e440245 | Ubuntu 12.10 cloudimg amd64   | ACTIVE |                                      |
| df8d56fc-9cea-4dfd-a8d3-28764de3cb08 | jenkins                       | ACTIVE |                                      |
+--------------------------------------+-------------------------------+--------+--------------------------------------+
```

The displayed image attributes are:

`ID`	Automatically generated UUID of the image
`Name`	Free form, human-readable name for image
`Status`	The status of the image. Images marked `ACTIVE` are available for use.
`Server`	For images that are created as snapshots of running instances, this is the UUID of the instance the snapshot derives from. For uploaded images, this field is blank.

Virtual hardware templates are called `flavors`. The default installation provides five flavors. By default, these are configurable by admin users, however that behavior can be changed by redefining the access controls for *`compute_extension:flavormanage`* in `/etc/nova/policy.json` on the `compute-api` server.

For a list of flavors that are available on your system:

```
$ nova flavor-list
+----+-----------+-----------+------+-----------+------+-------+-------------+
| ID |    Name   | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |
+----+-----------+-----------+------+-----------+------+-------+-------------+
| 1  | m1.tiny   | 512       | 1    | N/A       | 0    | 1     |             |
| 2  | m1.small  | 2048      | 20   | N/A       | 0    | 1     |             |
| 3  | m1.medium | 4096      | 40   | N/A       | 0    | 2     |             |
| 4  | m1.large  | 8192      | 80   | N/A       | 0    | 4     |             |
| 5  | m1.xlarge | 16384     | 160  | N/A       | 0    | 8     |             |
+----+-----------+-----------+------+-----------+------+-------+-------------+
```

Flavors

Admin users can use the **nova flavor-** commands to customize and manage flavors. To see the available flavor-related commands, run:

```
$ nova help | grep flavor-
   flavor-access-add   Add flavor access for the given tenant.
    flavor-access-list  Print access information about the given flavor.
    flavor-access-remove
                        Remove flavor access for the given tenant.
    flavor-create       Create a new flavor
    flavor-delete       Delete a specific flavor
    flavor-key          Set or unset extra_spec for a flavor.
    flavor-list         Print a list of available 'flavors' (sizes of
    flavor-show         Show details about the given flavor.
```

Note

- Configuration rights can be delegated to additional users by redefining the access controls for `compute_extension:flavormanage` in `/etc/nova/policy.json` on the `nova-api` server.
- To modify an existing flavor in the dashboard, you must delete the flavor and create a modified one with the same name.

Flavors define these elements:

Table 4.1. Identity Service configuration file sections

Element	Description
`Name`	A descriptive name. *XX.SIZE_NAME* is typically not required, though some third party tools may rely on it.
`Memory_MB`	Virtual machine memory in megabytes.
`Disk`	Virtual root disk size in gigabytes. This is an ephemeral disk that the base image is copied into. When booting from a persistent volume it is not used. The "0" size is a special case which uses the native base image size as the size of the ephemeral root volume.
`Ephemeral`	Specifies the size of a secondary ephemeral data disk. This is an empty, unformatted disk and exists only for the life of the instance.
`Swap`	Optional swap space allocation for the instance.
`VCPUs`	Number of virtual CPUs presented to the instance.
`RXTX_Factor`	Optional property allows created servers to have a different bandwidth cap than that defined in the network they are attached to. This factor is multiplied by the rxtx_base property of the network. Default value is 1.0. That is, the same as attached network.
`Is_Public`	Boolean value, whether flavor is available to all users or private to the tenant it was created in. Defaults to True.
`extra_specs`	Key and value pairs that define on which compute nodes a flavor can run. These pairs must match corresponding pairs on the compute nodes. Use to implement special resources, such as flavors that run on only compute nodes with GPU hardware.

Flavor customization can be limited by the hypervisor in use. For example the `libvirt` driver enables quotas on CPUs available to a VM, disk tuning, bandwidth I/O, watchdog behavior, random number generator device control, and instance VIF traffic control.

CPU limits

You can configure the CPU limits with control parameters with the **nova** client. For example, to configure the I/O limit, use:

```
$ nova flavor-key m1.small set
 quota:read_bytes_sec=10240000
$ nova flavor-key m1.small set
 quota:write_bytes_sec=10240000
```

There are optional CPU control parameters for weight shares, enforcement intervals for runtime quotas, and a quota for maximum allowed bandwidth:

- `cpu_shares` specifies the proportional weighted share for the domain. If this element is omitted, the service defaults to the OS provided defaults. There is no unit for the value; it is a relative measure based on the setting of other VMs. For example, a VM configured with value 2048 gets twice as much CPU time as a VM configured with value 1024.
- `cpu_period` specifies the enforcement interval (unit: microseconds) for QEMU and LXC hypervisors. Within a period, each VCPU of the domain is not allowed to consume more than the quota worth of runtime. The value should be in range `[1000, 1000000]`. A period with value 0 means no value.
- `cpu_quota` specifies the maximum allowed bandwidth (unit: microseconds). A domain with a negative-value quota indicates that the domain has infinite bandwidth, which means that it is not bandwidth controlled. The value should be in range `[1000, 18446744073709551]` or less than 0. A quota with value 0 means no value. You can use this feature to ensure that all vCPUs run at the same speed. For example:

  ```
  $ nova flavor-key m1.low_cpu set
   quota:cpu_quota=10000
  $ nova flavor-key m1.low_cpu set
   quota:cpu_period=20000
  ```

  In this example, the instance of `m1.low_cpu` can only consume a maximum of 50% CPU of a physical CPU computing capability.

Disk tuning

Using disk I/O quotas, you can set maximum disk write to 10 MB per second for a VM user. For example:

```
$ nova flavor-key m1.medium set
 disk_write_bytes_sec=10485760
```

The disk I/O options are:

- disk_read_bytes_sec
- disk_read_iops_sec
- disk_write_bytes_sec
- disk_write_iops_sec

	• disk_total_bytes_sec • disk_total_iops_sec The vif I/O options are: • vif_inbound_ average • vif_inbound_burst • vif_inbound_peak • vif_outbound_ average • vif_outbound_burst • vif_outbound_peak

Bandwidth I/O	Incoming and outgoing traffic can be shaped independently. The bandwidth element can have at most one inbound and at most one outbound child element. If you leave any of these children element out, no quality of service (QoS) is applied on that traffic direction. So, if you want to shape only the network's incoming traffic, use inbound only (and vice versa). Each element has one mandatory attribute average, which specifies the average bit rate on the interface being shaped.
	
	There are also two optional attributes (integer): `peak`, which specifies maximum rate at which bridge can send data (kilobytes/second), and `burst`, the amount of bytes that can be burst at peak speed (kilobytes). The rate is shared equally within domains connected to the network.
	
	The following example configures a bandwidth limit for instance network traffic:
	
	`$ nova flavor-key m1.small set quota:inbound_average=10240`
	`$ nova flavor-key m1.small set quota:outbound_average=10240`

Watchdog behavior	For the `libvirt` driver, you can enable and set the behavior of a virtual hardware watchdog device for each flavor. Watchdog devices keep an eye on the guest server, and carry out the configured action if the server hangs. The watchdog uses the i6300esb device (emulating a PCI Intel 6300ESB). If `hw_watchdog_action` is not specified, the watchdog is disabled.

To set the behavior, use:

```
$ nova flavor-key FLAVOR-NAME set
 hw_watchdog_action=ACTION
```

Valid *ACTION* values are:

- `disabled`—(default) The device is not attached.
- `reset`—Forcefully reset the guest.
- `poweroff`—Forcefully power off the guest.
- `pause`—Pause the guest.
- `none`—Only enable the watchdog; do nothing if the server hangs.

Note

Watchdog behavior set using a specific image's properties will override behavior set using flavors.

Random-number generator

If a random-number generator device has been added to the instance through its image properties, the device can be enabled and configured using:

```
$ nova flavor-key FLAVOR-NAME set
 hw_rng:allowed=True
$ nova flavor-key FLAVOR-NAME set
 hw_rng:rate_bytes=RATE-BYTES
$ nova flavor-key FLAVOR-NAME set
 hw_rng:rate_period=RATE-PERIOD
```

Where:

- *RATE-BYTES*—(Integer) Allowed amount of bytes that the guest can read from the host's entropy per period.
- *RATE-PERIOD*—(Integer) Duration of the read period in seconds.

Instance VIF traffic control

Flavors can also be assigned to particular projects. By default, a flavor is public and available to all projects. Private flavors are only accessible to those on the access list and are invisible to other projects. To create and assign a private flavor to a project, run these commands:

```
$ nova flavor-create --is-public false p1.medium
 auto 512 40 4
```

```
$ nova flavor-access-add 259d06a0-
ba6d-4e60-b42d-ab3144411d58
 86f94150ed744e08be565c2ff608eef9
```

Admin password injection

You can configure Compute to generate a random administrator (root) password and inject that password into the instance. If this feature is enabled, a user can **ssh** to an instance without an **ssh** keypair. The random password appears in the output of the **nova boot** command. You can also view and set the `admin` password from the dashboard.

Dashboard

The dashboard is configured by default to display the `admin` password and allow the user to modify it.

If you do not want to support password injection, we recommend disabling the password fields by editing your Dashboard `local_settings` file (file location will vary by Linux distribution, on Fedora/RHEL/CentOS: `/etc/openstack-dashboard/local_settings`, on Ubuntu and Debian: `/etc/openstack-dashboard/local_settings.py` and on openSUSE and SUSE Linux Enterprise Server: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`)

```
OPENSTACK_HYPERVISOR_FEATURE = {
...
    'can_set_password': False,
}
```

Libvirt-based hypervisors (KVM, QEMU, LXC)

For hypervisors such as KVM that use the libvirt backend, `admin` password injection is disabled by default. To enable it, set the following option in `/etc/nova/nova.conf`:

```
[libvirt]
inject_password=true
```

When enabled, Compute will modify the password of the root account by editing the `/etc/shadow` file inside of the virtual machine instance.

Note

Users can only ssh to the instance by using the admin password if:

- The virtual machine image is a Linux distribution
- The virtual machine has been configured to allow users to **ssh** as the root user. This is not the case for Ubuntu cloud images, which disallow **ssh** to the root account by default.

XenAPI (XenServer/XCP)

Compute uses the XenAPI agent to inject passwords into guests when using the XenAPI hypervisor backend. The virtual-machine image must be configured with the agent for password injection to work.

Windows images (all hypervisors)

To support the `admin` password for Windows virtual machines, you must configure the Windows image to retrieve the `admin` password on boot by installing an agent such as cloudbase-init.

Volumes

Depending on the setup of your cloud provider, they may give you an endpoint to use to manage volumes, or there may be an extension under the covers. In either case, you can use the **nova** CLI to manage volumes:

```
    volume-attach              Attach a volume to a server.
    volume-create              Add a new volume.
    volume-delete              Remove a volume.
    volume-detach              Detach a volume from a server.
    volume-list                List all the volumes.
    volume-show                Show details about a volume.
    volume-snapshot-create     Add a new snapshot.
    volume-snapshot-delete     Remove a snapshot.
    volume-snapshot-list       List all the snapshots.
    volume-snapshot-show       Show details about a snapshot.
    volume-type-create         Create a new volume type.
    volume-type-delete         Delete a specific flavor
    volume-type-list           Print a list of available 'volume types'.
    volume-update              Update an attached volume.
```

For example, to list IDs and names of Compute volumes, run:

```
$ nova volume-list
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
| ID                                   | Status    | Display Name | Size |
 Volume Type | Attached to |
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
| 1af4cb93-d4c6-4ee3-89a0-4b7885a3337e | available | PerfBlock    | 1    |
 Performance |             |
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
```

Networking with nova-network

Understanding the networking configuration options helps you design the best configuration for your Compute instances.

You can choose to either install and configure `nova-network` for networking between VMs or use the OpenStack Networking service (neutron) for networking. To configure Compute networking options with OpenStack Networking, see the Chapter 7, “Networking” [167].

Networking concepts

This section offers a brief overview of networking concepts for Compute.

Compute assigns a private IP address to each VM instance. (Currently, Compute with `nova-network` only supports Linux bridge networking that enables the virtual interfaces to connect to the outside network through the physical interface.) Compute makes a distinction between *fixed IPs* and *floating IPs*. Fixed IPs are IP addresses that are assigned to an instance on creation and stay the same until the instance is explicitly terminated. By contrast, floating IPs are addresses that can be dynamically associated with an instance. A floating IP address can be disassociated and associated with another instance at any time. A user can reserve a floating IP for their project.

The network controller with `nova-network` provides virtual networks to enable compute servers to interact with each other and with the public network. Compute with `nova-network` supports the following network modes, which are implemented as “Network Manager” types.

Flat Network Manager

In **Flat** mode, a network administrator specifies a subnet. IP addresses for VM instances are assigned from the subnet, and then injected into the image on launch. Each instance receives a fixed IP address from the pool of available addresses. A system administrator must create the Linux networking bridge (typically named `br100`, although this is configurable) on the systems running the `nova-network` service. All instances of the system are attached to the same bridge, and this is configured manually by the network administrator.

Note

Configuration injection currently only works on Linux-style systems that keep networking configuration in `/etc/network/interfaces`.

Flat DHCP Network Manager

In **FlatDHCP** mode, OpenStack starts a DHCP server (`dnsmasq`) to allocate IP addresses to VM instances from the specified subnet, in addition to manually configuring the networking bridge. IP addresses for VM instances are assigned from a subnet specified by the network administrator.

Like Flat Mode, all instances are attached to a single bridge on the compute node. Additionally, a DHCP server is running to configure instances (depending on single-/multi-host mode, alongside each `nova-network`). In this mode, Compute does a bit more configuration in that it attempts to bridge into an ethernet device (`flat_interface`, eth0 by default). For every instance, Compute allocates a fixed IP address and configures dnsmasq with the MAC/IP pair for the VM. Dnsmasq does not take part in the IP address allocation process, it only hands out IPs according to the mapping done by Compute. Instances receive their fixed IPs by doing a **dhcpdiscover**. These IPs are *not* assigned to any of the host's network interfaces, only to the VM's guest-side interface.

In any setup with flat networking, the hosts providing the `nova-network` service are responsible for forwarding traffic from the private network. They also run and configure `dnsmasq` as a DHCP server listening on this bridge, usually on IP address 10.0.0.1 (see DHCP server: dnsmasq). Compute can determine the NAT entries for each network, although sometimes NAT is not used, such as when configured with all public IPs or a hardware router is used (one of the HA options). Such hosts need to have `br100` configured and physically connected to any other nodes that are hosting VMs. You must set the `flat_network_bridge` option or create networks with the bridge parameter in order to avoid raising an error. Compute nodes have iptables/ebtables entries created for each project and instance to protect against IP/MAC address spoofing and ARP poisoning.

Note

In single-host Flat DHCP mode you *will* be able to ping VMs through their fixed IP from the `nova-network` node, but you *cannot* ping them from the compute nodes. This is expected behavior.

VLAN Network Manager

VLANManager mode is the default mode for OpenStack Compute. In this mode, Compute creates a VLAN and bridge for each tenant. For multiple-machine installation, the VLAN Network Mode requires a switch that supports VLAN tagging (IEEE 802.1Q). The tenant gets a range of private IPs that are only accessible from inside the VLAN. In order for a user to access the instances in their tenant, a special VPN instance (code named cloudpipe) needs to be created. Compute generates a certificate and key for the user to access the VPN and starts the VPN automatically. It provides a private network segment for each tenant's instances that can be accessed through a dedicated VPN connection from the Internet. In this mode, each tenant gets its own VLAN, Linux networking bridge, and subnet.

The subnets are specified by the network administrator, and are assigned dynamically to a tenant when required. A DHCP Server is started for each VLAN to pass out IP addresses to VM instances from the subnet assigned to the tenant. All instances belonging to one tenant are bridged into the same VLAN for that tenant.

OpenStack Compute creates the Linux networking bridges and VLANs when required.

These network managers can co-exist in a cloud system. However, because you cannot select the type of network for a given tenant, you cannot configure multiple network types in a single Compute installation.

All network managers configure the network using *network drivers*. For example, the Linux L3 driver (`l3.py` and `linux_net.py`), which makes use of `iptables`, `route` and other network management facilities, and libvirt's network filtering facilities. The driver is not tied to any particular network manager; all network managers use the same driver. The driver usually initializes (creates bridges and so on) only when the first VM lands on this host node.

All network managers operate in either *single-host* or *multi-host* mode. This choice greatly influences the network configuration. In single-host mode, a single `nova-network` service provides a default gateway for VMs and hosts a single DHCP server (`dnsmasq`). In multi-host mode, each compute node runs its own `nova-network` service. In both cases, all traffic between VMs and the outer world flows through `nova-network`. Each mode has its pros and cons (see the *Network Topology* section in the *OpenStack Operations Guide*.

Note

All networking options require network connectivity to be already set up between OpenStack physical nodes. OpenStack does not configure any physical network interfaces. All network managers automatically create VM virtual interfaces. Some, but not all, managers create network bridges such as `br100`.

All machines must have a *public* and *internal* network interface (controlled by the options: `public_interface` for the public interface, and `flat_interface` and `vlan_interface` for the internal interface with flat / VLAN managers). This guide refers to the public network as the external network and the private network as the internal or tenant network.

The internal network interface is used for communication with VMs; the interface should not have an IP address attached to it before OpenStack installation (it serves merely as a fabric where the actual endpoints are VMs and dnsmasq). Also, you must put the internal network interface in *promiscuous mode*, because it must receive packets whose target MAC address is of the guest VM, not of the host.

Throughout this documentation, the public network is sometimes referred to as the external network, while the internal network is also sometimes referred to as the private network or tenant network.

For flat and flat DHCP modes, use the following command to create a network:

```
$ nova network-create vmnet \
  --fixed-range-v4=10.0.0.0/24 --fixed-cidr=10.20.0.0/16 --bridge=br100
```

Where:

- `--fixed-range-v4-` specifies the network subnet.

- `--fixed-cidr` specifies a range of fixed IP addresses to allocate, and can be a subset of the `--fixed-range-v4` argument.
- `--bridge` specifies the bridge device to which this network is connected on every compute node.

DHCP server: dnsmasq

The Compute service uses dnsmasq as the DHCP server when running with either that Flat DHCP Network Manager or the VLAN Network Manager. The `nova-network` service is responsible for starting up `dnsmasq` processes.

The behavior of `dnsmasq` can be customized by creating a `dnsmasq` configuration file. Specify the configuration file using the `dnsmasq_config_file` configuration option. For example:

```
dnsmasq_config_file=/etc/dnsmasq-nova.conf
```

For an example of how to change the behavior of `dnsmasq` using a `dnsmasq` configuration file, see the *OpenStack Configuration Reference*. The `dnsmasq` documentation also has a more comprehensive dnsmasq configuration file example.

`dnsmasq` also acts as a caching DNS server for instances. You can explicitly specify the DNS server that `dnsmasq` should use by setting the `dns_server` configuration option in `/etc/nova/nova.conf`. The following example would configure `dnsmasq` to use Google's public DNS server:

```
dns_server=8.8.8.8
```

Logging output for `dnsmasq` goes to the `syslog` (typically `/var/log/syslog` or `/var/log/messages`, depending on Linux distribution). `dnsmasq` logging output can be useful for troubleshooting if VM instances boot successfully but are not reachable over the network.

A network administrator can run `nova-manage fixed reserve --address=`*`x.x.x.x`* to specify the starting point IP address (x.x.x.x) to reserve with the DHCP server. This reservation only affects which IP address the VMs start at, not the fixed IP addresses that the `nova-network` service places on the bridges.

Configure Compute to use IPv6 addresses

If you are using OpenStack Compute with `nova-network`, you can put Compute into IPv4/IPv6 dual-stack mode, so that it uses both IPv4 and IPv6 addresses for communication. In IPv4/IPv6 dual-stack mode, instances can acquire their IPv6 global unicast address by using a stateless address auto configuration mechanism [RFC 4862/2462]. IPv4/IPv6 dual-stack mode works with both `VlanManager` and `FlatDHCPManager` networking modes. In `VlanManager`, each project uses a different 64-bit global routing prefix. In `FlatDHCPManager`, all instances use one 64-bit global routing prefix.

This configuration was tested with VM images that have an IPv6 stateless address auto configuration capability. This capability is required for any VM you want to run with an IPv6 address. You must use EUI-64 address for stateless address auto configuration. Each node that executes a `nova-*` service must have `python-netaddr` and `radvd` installed.

Procedure 4.2. Switch into IPv4/IPv6 dual-stack mode

1. On all nodes running a `nova-*` service, install `python-netaddr`:

   ```
   # apt-get install python-netaddr
   ```

2. On all `nova-network` nodes, install `radvd` and configure IPv6 networking:

   ```
   # apt-get install radvd
   # echo 1 > /proc/sys/net/ipv6/conf/all/forwarding
   # echo 0 > /proc/sys/net/ipv6/conf/all/accept_ra
   ```

3. Edit the `nova.conf` file on all nodes to specify `use_ipv6 = True`.

4. Restart all `nova-*` services.

Note

You can add a fixed range for IPv6 addresses to the **nova network-create** command. Specify `public` or `private` after the `network-create` parameter.

```
$ nova network-create public --fixed-range-v4 fixed_range_v4 --
vlan vlan_id --vpn vpn_start --fixed-range-v6 fixed_range_v6
```

You can set IPv6 global routing prefix by using the `--fixed_range_v6` parameter. The default value for the parameter is: `fd00::/48`.

- When you use `FlatDHCPManager`, the command uses the original `--fixed_range_v6` value. For example:

  ```
  $ nova network-create public  --fixed-range-v4 10.0.2.0/24 --
  fixed-range-v6 fd00:1::/48
  ```

- When you use `VlanManager`, the command increments the subnet ID to create subnet prefixes. Guest VMs use this prefix to generate their IPv6 global unicast address. For example:

  ```
  $ nova network-create public --fixed-range-v4 10.0.1.0/24 --vlan
   100 --vpn 1000 --fixed-range-v6 fd00:1::/48
  ```

Table 4.2. Description of configuration options for ipv6

Configuration option = Default value	Description
[DEFAULT]	
fixed_range_v6 = fd00::/48	(StrOpt) Fixed IPv6 address block
gateway_v6 = None	(StrOpt) Default IPv6 gateway
ipv6_backend = rfc2462	(StrOpt) Backend to use for IPv6 generation
use_ipv6 = False	(BoolOpt) Use IPv6

Metadata service

Introduction

The Compute service uses a special metadata service to enable virtual machine instances to retrieve instance-specific data. Instances access the metadata service at `http://169.254.169.254`. The metadata service supports two sets of APIs: an OpenStack metadata API and an EC2-compatible API. Each of the APIs is versioned by date.

To retrieve a list of supported versions for the OpenStack metadata API, make a GET request to `http://169.254.169.254/openstack` For example:

```
$ curl http://169.254.169.254/openstack
2012-08-10
latest
```

To list supported versions for the EC2-compatible metadata API, make a GET request to `http://169.254.169.254`.

For example:

```
$ curl http://169.254.169.254
1.0
2007-01-19
2007-03-01
2007-08-29
2007-10-10
2007-12-15
2008-02-01
2008-09-01
2009-04-04
latest
```

If you write a consumer for one of these APIs, always attempt to access the most recent API version supported by your consumer first, then fall back to an earlier version if the most recent one is not available.

OpenStack metadata API

Metadata from the OpenStack API is distributed in JSON format. To retrieve the metadata, make a GET request to `http://169.254.169.254/openstack/2012-08-10/meta_data.json`.

For example:

```
$ curl http://169.254.169.254/openstack/2012-08-10/meta_data.json
```

```
{
   "uuid":"d8e02d56-2648-49a3-bf97-6be8f1204f38",
   "availability_zone":"nova",
   "hostname":"test.novalocal",
   "launch_index":0,
   "meta":{
      "priority":"low",
      "role":"webserver"
   },
   "public_keys":{
      "mykey":"ssh-rsa
 AAAAB3NzaC1yc2EAAAADAQABAAAAgQDYVEprvtYJXVOBN0XNKVVRNCRX6BlnNbI
+USLGais1sUWPwtSg7z9K9vhbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq8Oh5b2i71/
3BISpyxTBH/uZDHdslW2a+SrPDCeuMMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated
 by Nova\n"
   },
   "name":"test"
}
```

Instances also retrieve user data (passed as the `user_data` parameter in the API call or by the `--user_data` flag in the **nova boot** command) through the metadata service, by making a GET request to `http://169.254.169.254/openstack/2012-08-10/user_data`.

For example:

```
$ curl http://169.254.169.254/openstack/2012-08-10/user_data
                        #!/bin/bash
echo 'Extra user data here'
```

EC2 metadata API

The metadata service has an API that is compatible with version 2009-04-04 of the Amazon EC2 metadata service; virtual machine images that are designed for EC2 work properly with OpenStack.

The EC2 API exposes a separate URL for each metadata. You can retrieve a listing of these elements by making a GET query to `http://169.254.169.254/2009-04-04/meta-data/`

For example:

```
$ curl http://169.254.169.254/2009-04-04/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
kernel-id
local-hostname
local-ipv4
placement/
public-hostname
public-ipv4
public-keys/
ramdisk-id
reservation-id
security-groups
```

```
$ curl http://169.254.169.254/2009-04-04/meta-data/block-device-mapping/
ami
```

```
$ curl http://169.254.169.254/2009-04-04/meta-data/placement/
availability-zone
```

```
$ curl http://169.254.169.254/2009-04-04/meta-data/public-keys/
0=mykey
```

Instances can retrieve the public SSH key (identified by keypair name when a user requests a new instance) by making a GET request to `http://169.254.169.254/2009-04-04/meta-data/public-keys/0/openssh-key`.

For example:

```
$ curl http://169.254.169.254/2009-04-04/meta-data/public-keys/0/openssh-key
ssh-rsa AAAAB3NzaC1yc2EAAAADAQABAAAAgQDYVEprvtYJXVOBN0XNKVVRNCRX6BlnNbI
+USLGais1sUWPwtSg7z9K9vhbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq8Oh5b2i71/
3BISpyxTBH/uZDHdslW2a+SrPDCeuMMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated
 by Nova
```

Instances can retrieve user data by making a GET request to `http://169.254.169.254/2009-04-04/user-data`.

For example:

```
$ curl http://169.254.169.254/2009-04-04/user-data
#!/bin/bash
echo 'Extra user data here'
```

Run the metadata service

The metadata service is implemented by either the `nova-api` service or the `nova-api-metadata` service. (The `nova-api-metadata` service is generally only used when running in multi-host mode, it retrieves instance-specific metadata). If you are running the `nova-api` service, you must have `metadata` as one of the elements of the list of the `enabled_apis` configuration option in `/etc/nova/nova.conf`. The default `enabled_apis` configuration setting includes the metadata service, so you should not need to modify it.

Hosts access the service at `169.254.169.254:80`, and this is translated to `metadata_host:metadata_port` by an iptables rule established by the `nova-network` servce. In multi-host mode, you can set `metadata_host` to `127.0.0.1`.

To enable instances to reach the metadata service, the `nova-network` service configures iptables to NAT port `80` of the `169.254.169.254` address to the IP address specified in `metadata_host` (default `$my_ip`, which is the IP address of the `nova-network` service) and port specified in `metadata_port` (default `8775`) in `/etc/nova/nova.conf`.

Warning

The `metadata_host` configuration option must be an IP address, not a host name.

Note

The default Compute service settings assume that the `nova-network` service and the `nova-api` service are running on the same host. If this is not the case, you must make this change in the `/etc/nova/nova.conf` file on the host running the `nova-network` service:

Set the `metadata_host` configuration option to the IP address of the host where the `nova-api` service runs.

Table 4.3. Description of configuration options for metadata

Configuration option = Default value	Description
[DEFAULT]	
metadata_host = $my_ip	(StrOpt) The IP address for the metadata API server

Configuration option = Default value	Description
metadata_listen = 0.0.0.0	(StrOpt) The IP address on which the metadata API will listen.
metadata_listen_port = 8775	(IntOpt) The port on which the metadata API will listen.
metadata_manager = nova.api.manager.MetadataManager	(StrOpt) OpenStack metadata service manager
metadata_port = 8775	(IntOpt) The port for the metadata API port
metadata_workers = None	(IntOpt) Number of workers for metadata service. The default will be the number of CPUs available.
vendordata_driver = nova.api.metadata.vendordata_json.JsonFileVendorData	(StrOpt) Driver to use for vendor data
vendordata_jsonfile_path = None	(StrOpt) File to load json formatted vendor data from

Enable ping and SSH on VMs

Be sure you enable access to your VMs by using the **euca-authorize** or **nova secgroup-add-rule** command. These commands enable you to **ping** and **ssh** to your VMs:

Note

You must run these commands as root only if the credentials used to interact with `nova-api` are in `/root/.bashrc`. If the EC2 credentials are the `.bashrc` file for another user, you must run these commands as the user.

Run **nova** commands:

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

Using euca2ools:

```
$ euca-authorize -P icmp -t -1:-1 -s 0.0.0.0/0 default
$ euca-authorize -P tcp -p 22 -s 0.0.0.0/0 default
```

If you still cannot ping or SSH your instances after issuing the **nova secgroup-add-rule** commands, look at the number of `dnsmasq` processes that are running. If you have a running instance, check to see that TWO `dnsmasq` processes are running. If not, perform the following commands as root:

```
# killall dnsmasq
# service nova-network restart
```

Configure public (floating) IP addresses

If you are using Compute's `nova-network` instead of OpenStack Networking (neutron) for networking in OpenStack, use procedures in this section to configure floating IP addresses. For instructions on how to configure OpenStack Networking (neutron) to provide access to instances through floating IP addresses, see the section called “L3 routing and NAT” [238].

Private and public IP addresses

Every virtual instance is automatically assigned a private IP address. You can optionally assign public IP addresses to instances. The term *floating IP* refers to an IP address, typically public, that you can dynamically add to a running virtual instance. OpenStack Compute uses Network Address Translation (NAT) to assign floating IPs to virtual instances.

If you plan to use this feature, you must add edit the `/etc/nova/nova.conf` file to specify to which interface the `nova-network` service binds public IP addresses, as follows:

```
public_interface=vlan100
```

If you make changes to the `/etc/nova/nova.conf` file while the `nova-network` service is running, you must restart the service.

Traffic between VMs using floating IPs

Because floating IPs are implemented by using a source NAT (SNAT rule in iptables), security groups can display inconsistent behavior if VMs use their floating IP to communicate with other VMs, particularly on the same physical host. Traffic from VM to VM across the fixed network does not have this issue, and so this is the recommended path. To ensure that traffic does not get SNATed to the floating range, explicitly set:

```
dmz_cidr=x.x.x.x/y
```

The `x.x.x.x/y` value specifies the range of floating IPs for each pool of floating IPs that you define. If the VMs in the source group have floating IPs, this configuration is also required.

Enable IP forwarding

By default, IP forwarding is disabled on most Linux distributions. To use the floating IP feature, you must enable IP forwarding.

Note

You must enable IP forwarding only on the nodes that run the `nova-network` service. If you use `multi_host` mode, ensure that you enable it on all compute nodes. Otherwise, enable it on only the node that runs the `nova-network` service.

To check whether forwarding is enabled, run:

```
$ cat /proc/sys/net/ipv4/ip_forward
0
```

Alternatively, you can run:

```
$ sysctl net.ipv4.ip_forward
net.ipv4.ip_forward = 0
```

In the previous example, IP forwarding is **disabled**. To enable it dynamically, run:

```
# sysctl -w net.ipv4.ip_forward=1
```

Or:

```
# echo 1 > /proc/sys/net/ipv4/ip_forward
```

To make the changes permanent, edit the `/etc/sysctl.conf` file and update the IP forwarding setting:

```
net.ipv4.ip_forward = 1
```

Save the file and run the following command to apply the changes:

```
# sysctl -p
```

You can also update the setting by restarting the network service:

- On Ubuntu, run:

  ```
  #/etc/init.d/procps.sh restart
  ```

- On RHEL/Fedora/CentOS, run:

  ```
  # service network restart
  ```

Create a list of available floating IP addresses

Compute maintains a list of floating IP addresses that you can assign to instances. Use the **nova-manage floating create** command to add entries to this list.

For example:

```
# nova-manage floating create --pool=nova --ip_range=68.99.26.170/31
```

You can use the following **nova-manage** commands to perform floating IP operations:

-
  ```
  # nova-manage floating list
  ```

  Lists the floating IP addresses in the pool.

-
  ```
  # nova-manage floating create --pool=[pool name] --ip_range=[CIDR]
  ```

  Creates specific floating IPs for either a single address or a subnet.

-
  ```
  # nova-manage floating delete [CIDR]
  ```

  Removes floating IP addresses using the same parameters as the create command.

For information about how administrators can associate floating IPs with instances, see Manage IP addresses in the *OpenStack Admin User Guide*.

Automatically add floating IPs

You can configure the `nova-network` service to automatically allocate and assign a floating IP address to virtual instances when they are launched. Add the following line to the `/etc/nova/nova.conf` file and restart the `nova-network` service:

```
auto_assign_floating_ip=True
```

Note

If you enable this option and all floating IP addresses have already been allocated, the **nova boot** command fails.

Remove a network from a project

You cannot remove a network that has already been associated to a project by simply deleting it.

To determine the project ID, you must have administrative rights. You can disassociate the project from the network with a scrub command and the project ID as the final parameter:

```
# nova-manage project scrub --project=<id>
```

Multiple interfaces for your instances (multinic)

The multinic feature allows you to plug more than one interface to your instances, making it possible to make several use cases available:

- SSL Configurations (VIPs)
- Services failover/ HA
- Bandwidth Allocation
- Administrative/ Public access to your instances

Each VIF is representative of a separate network with its own IP block. Every network mode introduces its own set of changes regarding the multinic usage:

Figure 4.4. multinic flat manager

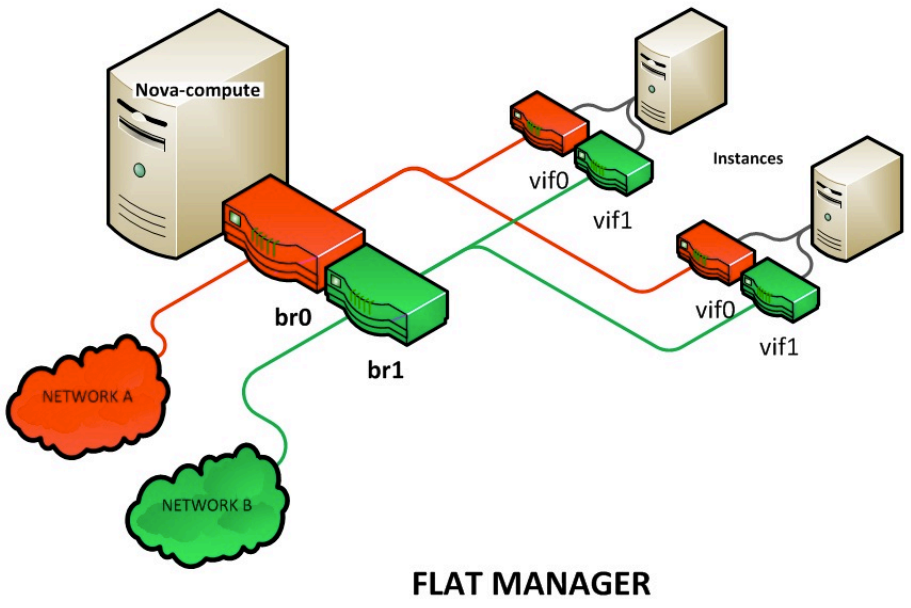


Figure 4.5. multinic flatdhcp manager

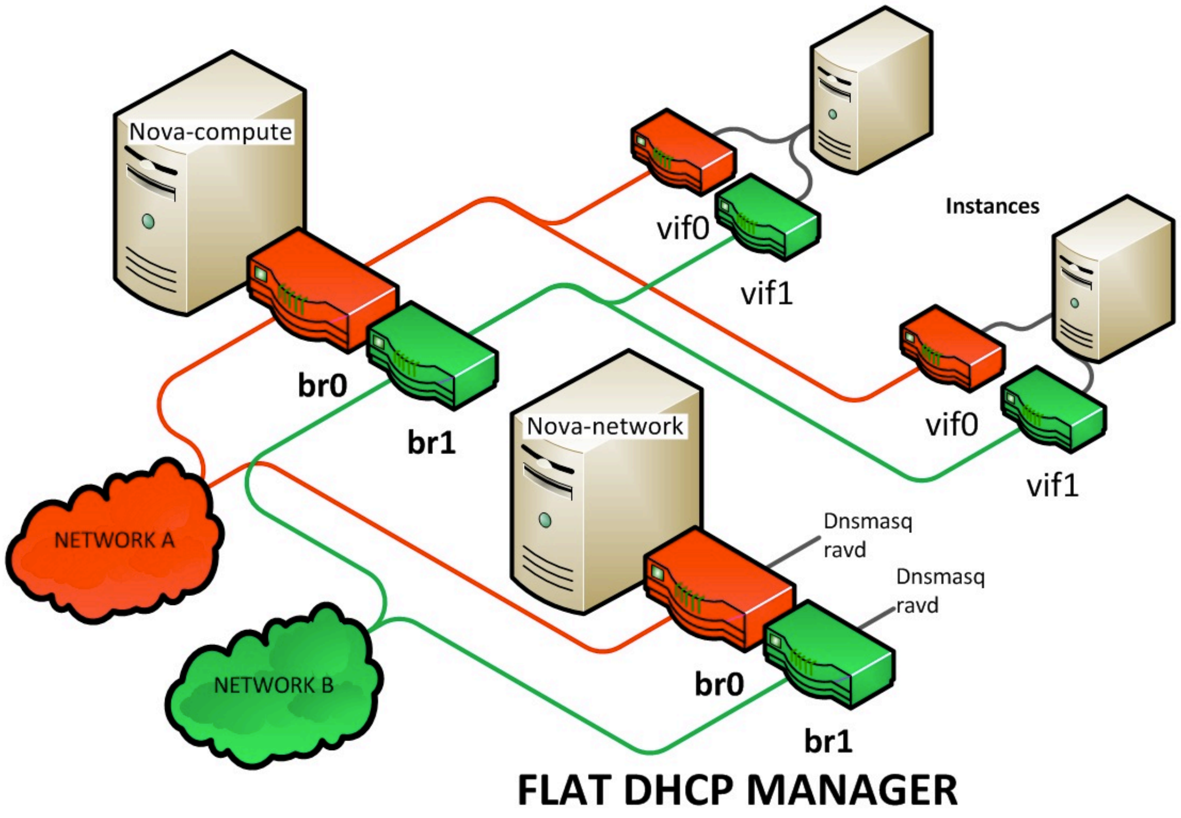
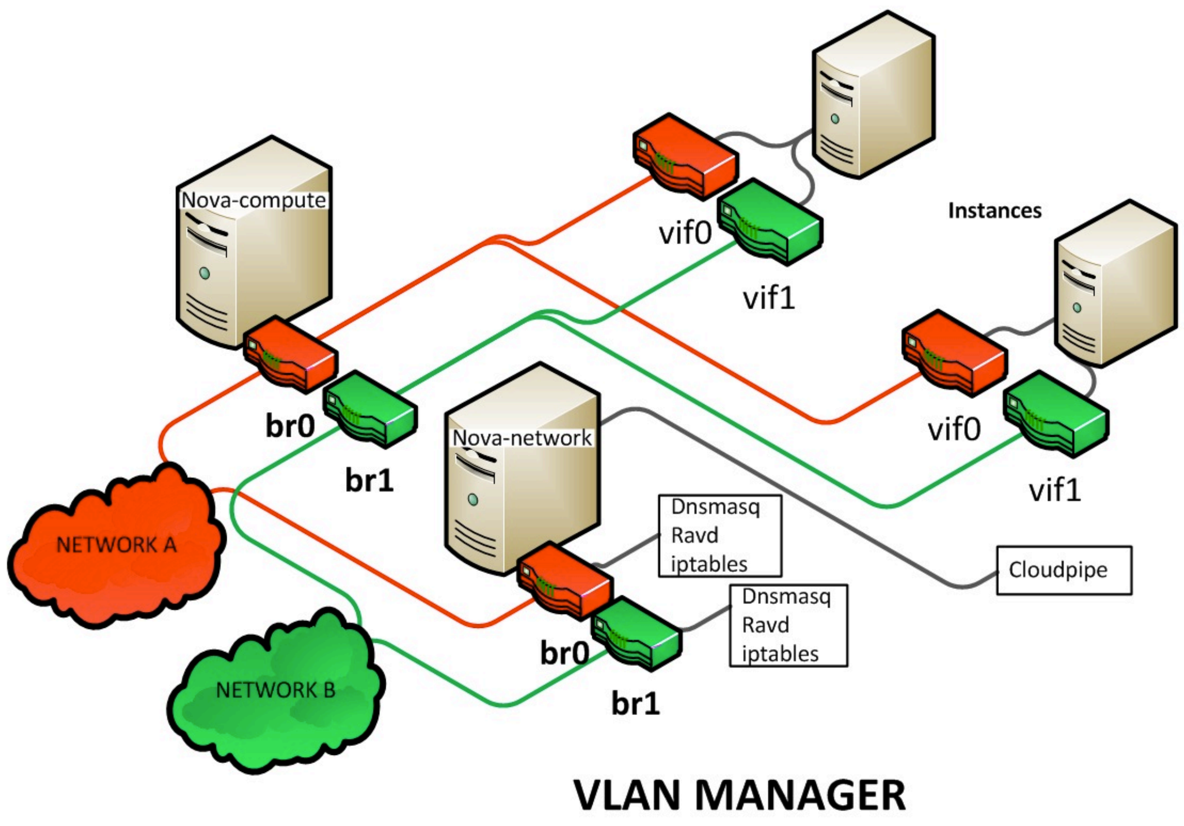


Figure 4.6. multinic VLAN manager



Use the multinic feature

In order to use the multinic feature, first create two networks, and attach them to your tenant (still named 'project' on the command line):

```
$ nova network-create first-net --fixed-range-v4=20.20.0.0/24 --project-id=
$your-project
$ nova network-create second-net --fixed-range-v4=20.20.10.0/24 --project-id=
$your-project
```

Now every time you spawn a new instance, it gets two IP addresses from the respective DHCP servers:

```
$ nova list
+-----+------------+--------+----------------------------------------+
 |  ID |    Name    | Status |                Networks                |
 +-----+------------+--------+----------------------------------------+
 | 124 | Server 124 | ACTIVE | network2=20.20.0.3; private=20.20.10.14|
 +-----+------------+--------+----------------------------------------+
```

Note

Make sure to power up the second interface on the instance, otherwise that last won't be reachable through its second IP. Here is an example of how to setup the interfaces within the instance (this is the configuration that needs to be applied inside the image):

`/etc/network/interfaces`

```
# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

Note

If the Virtual Network Service Neutron is installed, it is possible to specify the networks to attach to the respective interfaces by using the `--nic` flag when invoking the `nova` command:

```
$ nova boot --image ed8b2a37-5535-4a5f-a615-443513036d71 --flavor
 1 --nic net-id= <id of first network>  --nic net-id= <id of first
 network>  test-vm1
```

Troubleshoot Networking

Cannot reach floating IPs

If you cannot reach your instances through the floating IP address, check the following:

- Ensure the default security group allows ICMP (ping) and SSH (port 22), so that you can reach the instances:

```
$ nova secgroup-list-rules default
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port |  IP Range | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | -1      | 0.0.0.0/0 |              |
| tcp         | 22        | 22      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

- Ensure the NAT rules have been added to `iptables` on the node that `nova-network` is running on, as root:

```
# iptables -L -nv -t nat
-A nova-network-PREROUTING -d 68.99.26.170/32 -j DNAT --to-destination 10.0.
0.3
-A nova-network-floating-snat -s 10.0.0.3/32 -j SNAT --to-source 68.99.26.
170
```

- Check that the public address, in this example "68.99.26.170", has been added to your public interface. You should see the address in the listing when you enter "ip addr" at the command prompt.

```
$ ip addr
```

```
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP qlen
 1000
link/ether xx:xx:xx:17:4b:c2 brd ff:ff:ff:ff:ff:ff
inet 13.22.194.80/24 brd 13.22.194.255 scope global eth0
inet 68.99.26.170/32 scope global eth0
inet6 fe80::82b:2bf:fe1:4b2/64 scope link
valid_lft forever preferred_lft forever
```

Note that you cannot SSH to an instance with a public IP from within the same server as the routing configuration won't allow it.

- You can use **tcpdump** to identify if packets are being routed to the inbound interface on the compute host. If the packets are reaching the compute hosts but the connection is failing, the issue may be that the packet is being dropped by reverse path filtering. Try disabling reverse-path filtering on the inbound interface. For example, if the inbound interface is `eth2`, as root, run:

```
# sysctl -w net.ipv4.conf.eth2.rp_filter=0
```

If this solves your issue, add the following line to `/etc/sysctl.conf` so that the reverse-path filter is disabled the next time the compute host reboots:

```
net.ipv4.conf.rp_filter=0
```

Disable firewall

To help debug networking issues with reaching VMs, you can disable the firewall by setting the following option in `/etc/nova/nova.conf`:

```
firewall_driver=nova.virt.firewall.NoopFirewallDriver
```

We strongly recommend you remove this line to re-enable the firewall once your networking issues have been resolved.

Packet loss from instances to nova-network server (VLANManager mode)

If you can SSH to your instances but you find that the network interactions to your instance is slow, or if you find that running certain operations are slower than they should be (for example, **sudo**), then there may be packet loss occurring on the connection to the instance.

Packet loss can be caused by Linux networking configuration settings related to bridges. Certain settings can cause packets to be dropped between the VLAN interface (for example, `vlan100`) and the associated bridge interface (for example, `br100`) on the host running the `nova-network` service.

One way to check whether this is the issue in your setup, is to open up three terminals and run the following commands:

1. In the first terminal, on the host running nova-network, use **tcpdump** on the VLAN interface to monitor DNS-related traffic (UDP, port 53). As root, run:

```
# tcpdump -K -p -i vlan100 -v -vv udp port 53
```

2. In the second terminal, also on the host running nova-network, use **tcpdump** to monitor DNS-related traffic on the bridge interface. As root, run:

```
# tcpdump -K -p -i br100 -v -vv udp port 53
```

3. In the third terminal, SSH inside of the instance and generate DNS requests by using the **nslookup** command:

   ```
   $ nslookup www.google.com
   ```

   The symptoms may be intermittent, so try running **nslookup** multiple times. If the network configuration is correct, the command should return immediately each time. If it is not functioning properly, the command hangs for several seconds.

4. If the **nslookup** command sometimes hangs, and there are packets that appear in the first terminal but not the second, then the problem may be due to filtering done on the bridges. Try to disable filtering, run the following commands as root:

   ```
   # sysctl -w net.bridge.bridge-nf-call-arptables=0
   # sysctl -w net.bridge.bridge-nf-call-iptables=0
   # sysctl -w net.bridge.bridge-nf-call-ip6tables=0
   ```

   If this solves your issue, add the following line to `/etc/sysctl.conf` so that these changes take effect the next time the host reboots:

   ```
   net.bridge.bridge-nf-call-arptables=0
   net.bridge.bridge-nf-call-iptables=0
   net.bridge.bridge-nf-call-ip6tables=0
   ```

KVM: Network connectivity works initially, then fails

Some administrators have observed an issue with the KVM hypervisor where instances running Ubuntu 12.04 sometimes loses network connectivity after functioning properly for a period of time. Some users have reported success with loading the vhost_net kernel module as a workaround for this issue (see bug #997978) . This kernel module may also improve network performance on KVM. To load the kernel module, as root:

```
# modprobe vhost_net
```

Note

Loading the module has no effect on running instances.

System administration

By understanding how the different installed nodes interact with each other, you can administer the Compute installation. Compute offers many ways to install using multiple servers but the general idea is that you can have multiple compute nodes that control the virtual servers and a cloud controller node that contains the remaining Compute services.

The Compute cloud works through the interaction of a series of daemon processes named `nova-*` that reside persistently on the host machine or machines. These binaries can all run on the same machine or be spread out on multiple boxes in a large deployment. The responsibilities of services and drivers are:

- Services:

  - `nova-api`. Receives xml requests and sends them to the rest of the system. It is a wsgi app that routes and authenticate requests. It supports the EC2 and OpenStack APIs. There is a `nova-api.conf` file created when you install Compute.
  - `nova-cert`. Provides the certificate manager.
  - `nova-compute`. Responsible for managing virtual machines. It loads a Service object which exposes the public methods on ComputeManager through Remote Procedure Call (RPC).
  - `nova-conductor`. Provides database-access support for Compute nodes (thereby reducing security risks).
  - `nova-consoleauth`. Handles console authentication.
  - `nova-objectstore`: The `nova-objectstore` service is an ultra simple file-based storage system for images that replicates most of the S3 API. It can be replaced with OpenStack Image Service and a simple image manager or use OpenStack Object Storage as the virtual machine image storage facility. It must reside on the same node as `nova-compute`.
  - `nova-network`. Responsible for managing floating and fixed IPs, DHCP, bridging and VLANs. It loads a Service object which exposes the public methods on one of the subclasses of NetworkManager. Different networking strategies are available to the service by changing the network_manager configuration option to FlatManager, FlatDHCPManager, or VlanManager (default is VLAN if no other is specified).
  - `nova-scheduler`. Dispatches requests for new virtual machines to the correct node.
  - `nova-novncproxy`. Provides a VNC proxy for browsers (enabling VNC consoles to access virtual machines).
- Some services have drivers that change how the service implements the core of its functionality. For example, the `nova-compute` service supports drivers that let you choose with which hypervisor type it will talk. `nova-network` and `nova-scheduler` also have drivers.

Compute service architecture

The following basic categories describe the service architecture and what's going on within the cloud controller.

API server

At the heart of the cloud framework is an API server. This API server makes command and control of the hypervisor, storage, and networking programmatically available to users.

The API endpoints are basic HTTP web services which handle authentication, authorization, and basic command and control functions using various API interfaces under the Amazon, Rackspace, and related models. This enables API compatibility with multiple existing tool sets created for interaction with offerings from other vendors. This broad compatibility prevents vendor lock-in.

Message queue

A messaging queue brokers the interaction between compute nodes (processing), the networking controllers (software which controls network infrastructure), API endpoints, the scheduler (determines which physical hardware to allocate to a virtual resource), and similar components. Communication to and from the cloud controller is by HTTP requests through multiple API endpoints.

A typical message passing event begins with the API server receiving a request from a user. The API server authenticates the user and ensures that the user is permitted to issue the subject command. The availability of objects implicated in the request is evaluated and, if available, the request is routed to the queuing engine for the relevant workers. Workers continually listen to the queue based on their role, and occasionally their type host name. When an applicable work request arrives on the queue, the worker takes assignment of the task and begins its execution. Upon completion, a response is dispatched to the queue which is received by the API server and relayed to the originating user. Database entries are queried, added, or removed as necessary throughout the process.

Compute worker

Compute workers manage computing instances on host machines. The API dispatches commands to compute workers to complete these tasks:

- Run instances
- Terminate instances
- Reboot instances
- Attach volumes
- Detach volumes
- Get console output

Network Controller

The Network Controller manages the networking resources on host machines. The API server dispatches commands through the message queue, which are subsequently processed by Network Controllers. Specific operations include:

- Allocate fixed IP addresses
- Configuring VLANs for projects
- Configuring networks for compute nodes

Manage Compute users

Access to the Euca2ools (ec2) API is controlled by an access and secret key. The user’s access key needs to be included in the request, and the request must be signed with the secret key. Upon receipt of API requests, Compute verifies the signature and runs commands on behalf of the user.

To begin using Compute, you must create a user with the Identity Service.

Manage the cloud

A system administrator can use the **nova** client and the **Euca2ools** commands to manage the cloud.

Both nova client and euca2ools can be used by all users, though specific commands might be restricted by Role Based Access Control in the Identity Service.

Procedure 4.3. To use the nova client

1. Installing the python-novaclient package gives you a nova shell command that enables Compute API interactions from the command line. Install the client, and then provide your user name and password (typically set as environment variables for convenience), and then you have the ability to send commands to your cloud on the command line.

   To install python-novaclient, download the tarball from http://pypi.python.org/pypi/python-novaclient/2.6.3#downloads and then install it in your favorite python environment.

   ```
   $ curl -O http://pypi.python.org/packages/source/p/python-novaclient/
   python-novaclient-2.6.3.tar.gz
   $ tar -zxvf python-novaclient-2.6.3.tar.gz
   $ cd python-novaclient-2.6.3
   ```

   As root execute:

   ```
   # python setup.py install
   ```

2. Confirm the installation by running:

   ```
   $ nova help
   usage: nova [--version] [--debug] [--os-cache] [--timings]
               [--timeout <seconds>] [--os-username <auth-user-name>]
               [--os-password <auth-password>]
               [--os-tenant-name <auth-tenant-name>]
               [--os-tenant-id <auth-tenant-id>] [--os-auth-url <auth-url>]
               [--os-region-name <region-name>] [--os-auth-system <auth-
   system>]
               [--service-type <service-type>] [--service-name <service-
   name>]
               [--volume-service-name <volume-service-name>]
               [--endpoint-type <endpoint-type>]
               [--os-compute-api-version <compute-api-ver>]
               [--os-cacert <ca-certificate>] [--insecure]
               [--bypass-url <bypass-url>]
               <subcommand> ...
   ```

   Note

   This command returns a list of **nova** commands and parameters. To obtain help for a subcommand, run:

```
$ nova help subcommand
```

You can also refer to the *OpenStack Command-Line Reference* for a complete listing of **nova** commands and parameters.

3. Set the required parameters as environment variables to make running commands easier. For example, you can add `--os-username` as a **nova** option, or set it as an environment variable. To set the user name, password, and tenant as environment variables, use:

```
$ export OS_USERNAME=joecool
$ export OS_PASSWORD=coolword
$ export OS_TENANT_NAME=coolu
```

4. Using the Identity Service, you are supplied with an authentication endpoint, which Compute recognizes as the OS_AUTH_URL.

```
$ export OS_AUTH_URL=http://hostname:5000/v2.0
$ export NOVA_VERSION=1.1
```

Use the euca2ools commands

For a command-line interface to EC2 API calls, use the **euca2ools** command-line tool. See http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3

Show usage statistics for hosts and instances

You can show basic statistics on resource usage for hosts and instances.

Note

For more sophisticated monitoring, see the Ceilometer project, which is under development. You can also use tools, such as Ganglia or Graphite, to gather more detailed data.

Show host usage statistics

The following examples show the host usage statistics for a host called `devstack`.

- List the hosts and the nova-related services that run on them:

```
$ nova host-list
+-----------+-------------+----------+
| host_name | service     | zone     |
+-----------+-------------+----------+
| devstack  | conductor   | internal |
| devstack  | compute     | nova     |
| devstack  | cert        | internal |
| devstack  | network     | internal |
| devstack  | scheduler   | internal |
| devstack  | consoleauth | internal |
+-----------+-------------+----------+
```

- Get a summary of resource usage of all of the instances running on the host:

```
$ nova host-describe devstack
 +-----------+----------------------------------+-----+-----------+---------
+
| HOST     | PROJECT                          | cpu | memory_mb | disk_gb |
+----------+----------------------------------+-----+-----------+---------+
| devstack | (total)                          | 2   | 4003      | 157     |
| devstack | (used_now)                       | 3   | 5120      | 40      |
| devstack | (used_max)                       | 3   | 4608      | 40      |
| devstack | b70d90d65e464582b6b2161cf3603ced | 1   | 512       | 0       |
| devstack | 66265572db174a7aa66eba661f58eb9e | 2   | 4096      | 40      |
+----------+----------------------------------+-----+-----------+---------+
```

The `cpu` column shows the sum of the virtual CPUs for instances running on the host.

The `memory_mb` column shows the sum of the memory (in MB) allocated to the instances that run on the host.

The `disk_gb` column shows the sum of the root and ephemeral disk sizes (in GB) of the instances that run on the host.

The row that has the value `used_now` in the `PROJECT` column shows the sum of the resources allocated to the instances that run on the host, plus the resources allocated to the virtual machine of the host itself.

The row that has the value `used_max` row in the `PROJECT` column shows the sum of the resources allocated to the instances that run on the host.

Note

These values are computed by using information about the flavors of the instances that run on the hosts. This command does not query the CPU usage, memory usage, or hard disk usage of the physical host.

Show instance usage statistics

- Get CPU, memory, I/O, and network statistics for an instance.

1. List instances:

```
$ nova list
+--------------------------------------+----------------------+--------
+------------+-------------+------------------+
| ID                                   | Name                 | Status |
 Task State | Power State | Networks         |
+--------------------------------------+----------------------+--------
+------------+-------------+------------------+
| 84c6e57d-a6b1-44b6-81eb-fcb36afd31b5 | myCirrosServer       | ACTIVE |
 None       | Running     | private=10.0.0.3 |
| 8a99547e-7385-4ad1-ae50-4ecfaaad5f42 | myInstanceFromVolume | ACTIVE |
 None       | Running     | private=10.0.0.4 |
+--------------------------------------+----------------------+--------
+------------+-------------+------------------+
```

2. Get diagnostic statistics:

```
$ nova diagnostics myCirrosServer
```

```
+------------------+----------------+
| Property         | Value          |
+------------------+----------------+
| vnet1_rx         | 1210744        |
| cpu0_time        | 19624610000000 |
| vda_read         | 0              |
| vda_write        | 0              |
| vda_write_req    | 0              |
| vnet1_tx         | 863734         |
| vnet1_tx_errors  | 0              |
| vnet1_rx_drop    | 0              |
| vnet1_tx_packets | 3855           |
| vnet1_tx_drop    | 0              |
| vnet1_rx_errors  | 0              |
| memory           | 2097152        |
| vnet1_rx_packets | 5485           |
| vda_read_req     | 0              |
| vda_errors       | -1             |
+------------------+----------------+
```

- Get summary statistics for each tenant:

```
$ nova usage-list
Usage from 2013-06-25 to 2013-07-24:
+----------------------------------+-----------+--------------+-----------
+---------------+
| Tenant ID                        | Instances | RAM MB-Hours | CPU Hours |
 Disk GB-Hours |
+----------------------------------+-----------+--------------+-----------
+---------------+
| b70d90d65e464582b6b2161cf3603ced | 1         | 344064.44    | 672.00    |
 0.00          |
| 66265572db174a7aa66eba661f58eb9e | 3         | 671626.76    | 327.94    |
 6558.86       |
+----------------------------------+-----------+--------------+-----------
+---------------+
```

Manage logs

Logging module

To specify a configuration file to change the logging behavior, add this line to the /etc/nova/nova.conf file . To change the logging level, such as DEBUG, INFO, WARNING, ERROR), use:

```
log-config=/etc/nova/logging.conf
```

The logging configuration file is an ini-style configuration file, which must contain a section called logger_nova, which controls the behavior of the logging facility in the nova-* services. For example:

```
[logger_nova]
level = INFO
handlers = stderr
qualname = nova
```

This example sets the debugging level to INFO (which less verbose than the default DEBUG setting).

- For more details on the logging configuration syntax, including the meaning of the `handlers` and `quaname` variables, see the Python documentation on logging configuration file format f.
- For an example `logging.conf` file with various defined handlers, see the *OpenStack Configuration Reference*.

Syslog

You can configure OpenStack Compute services to send logging information to `syslog`. This is useful if you want to use `rsyslog`, which forwards the logs to a remote machine. You need to separately configure the Compute service (nova), the Identity service (keystone), the Image Service (glance), and, if you are using it, the Block Storage service (cinder) to send log messages to `syslog`. To do so, add the following lines to:

- `/etc/nova/nova.conf`
- `/etc/keystone/keystone.conf`
- `/etc/glance/glance-api.conf`
- `/etc/glance/glance-registry.conf`
- `/etc/cinder/cinder.conf`

```
verbose = False
debug = False
use_syslog = True
syslog_log_facility = LOG_LOCAL0
```

In addition to enabling `syslog`, these settings also turn off more verbose output and debugging output from the log.

Note

Although the example above uses the same local facility for each service (`LOG_LOCAL0`, which corresponds to `syslog` facility `LOCAL0`), we recommend that you configure a separate local facility for each service, as this provides better isolation and more flexibility. For example, you may want to capture logging information at different severity levels for different services. `syslog` allows you to define up to seven local facilities, `LOCAL0, LOCAL1, ..., LOCAL7`. For more details, see the `syslog` documentation.

Rsyslog

`rsyslog` is a useful tool for setting up a centralized log server across multiple machines. We briefly describe the configuration to set up an `rsyslog` server; a full treatment of `rsyslog` is beyond the scope of this document. We assume `rsyslog` has already been installed on your hosts (default for most Linux distributions).

This example provides a minimal configuration for `/etc/rsyslog.conf` on the log server host, which receives the log files:

```
# provides TCP syslog reception
$ModLoad imtcp
$InputTCPServerRun 1024
```

Add a filter rule to `/etc/rsyslog.conf` which looks for a host name. The example below uses *`compute-01`* as an example of a compute host name:

```
:hostname, isequal, "compute-01" /mnt/rsyslog/logs/compute-01.log
```

On each compute host, create a file named `/etc/rsyslog.d/60-nova.conf`, with the following content:

```
# prevent debug from dnsmasq with the daemon.none parameter
*.*;auth,authpriv.none,daemon.none,local0.none -/var/log/syslog
# Specify a log level of ERROR
local0.error    @@172.20.1.43:1024
```

Once you have created this file, restart your `rsyslog` daemon. Error-level log messages on the compute hosts should now be sent to your log server.

Secure with root wrappers

The root wrapper enables an unprivileged user to run a number of Compute actions as the root user in the safest manner possible. Historically, Compute used a specific `sudoers` file that listed every command that the Compute user was allowed to run, and used **sudo** to run that command as `root`. However this was difficult to maintain (the `sudoers` file was in packaging), and did not enable complex filtering of parameters (advanced filters). The rootwrap was designed to solve those issues.

How rootwrap works

Instead of calling **sudo make me a sandwich**, Compute services start with a **nova-rootwrap** call; for example, **sudo nova-rootwrap /etc/nova/rootwrap.conf make me a sandwich**. A generic sudoers entry lets the Compute user run **nova-rootwrap** as root. The **nova-rootwrap** code looks for filter definition directories in its configuration file, and loads command filters from them. Then it checks if the command requested by Compute matches one of those filters, in which case it executes the command (as root). If no filter matches, it denies the request.

Note

To use **nova-rootwrap**, you must be aware of the issues with using NFS and root-owned files. The NFS share must be configured with the `no_root_squash` option enabled.

Security model

The escalation path is fully controlled by the root user. A sudoers entry (owned by root) allows Compute to run (as root) a specific rootwrap executable, and only with a specific configuration file (which should be owned by root). **nova-rootwrap** imports the Python modules it needs from a cleaned (and system-default) *PYTHONPATH*. The configuration file (also root-owned) points to root-owned filter definition directories, which contain root-owned filters definition files. This chain ensures that the Compute user itself is not in control of the configuration or modules used by the **nova-rootwrap** executable.

Details of rootwrap.conf

You configure **nova-rootwrap** in the `rootwrap.conf` file. Because it's in the trusted security path, it must be owned and writable by only the root user. The file's location is specified both in the sudoers entry and in the `nova.conf` configuration file with the `rootwrap_config=entry`.

The `rootwrap.conf` file uses an INI file format with these sections and parameters:

Table 4.4. rootwrap.conf configuration options

Configuration option=Default value	(Type) Description
[DEFAULT] filters_path=/etc/nova/rootwrap.d,/usr/share/nova/rootwrap	(ListOpt) Comma-separated list of directories containing filter definition files. Defines where filters for root wrap are stored. Directories defined on this line should all exist, be owned and writable only by the root user.

Details of .filters files

Filters definition files contain lists of filters that **nova-rootwrap** will use to allow or deny a specific command. They are generally suffixed by .filters. Since they are in the trusted security path, they need to be owned and writable only by the root user. Their location is specified in the `rootwrap.conf` file.

Filter definition files use an INI file format with a [Filters] section and several lines, each with a unique parameter name (different for each filter that you define):

Table 4.5. .filters configuration options

Configuration option=Default value	(Type) Description
[Filters] filter_name=kpartx: CommandFilter, /sbin/kpartx, root	(ListOpt) Comma-separated list containing first the Filter class to use, followed by that Filter arguments (which vary depending on the Filter class selected).

Configure migrations

Note

Only cloud administrators can perform live migrations. If your cloud is configured to use cells, you can perform live migration within but not between cells.

Migration enables an administrator to move a virtual-machine instance from one compute host to another. This feature is useful when a compute host requires maintenance. Migration can also be useful to redistribute the load when many VM instances are running on a specific physical machine.

The migration types are:

- **Migration** (or non-live migration). The instance is shut down (and the instance knows that it was rebooted) for a period of time to be moved to another hypervisor.
- **Live migration** (or true live migration). Almost no instance downtime. Useful when the instances must be kept running during the migration. The types of *live migration* are:

- **Shared storage-based live migration**. Both hypervisors have access to shared storage.
- **Block live migration**. No shared storage is required. Incompatible with read-only devices such as CD-ROMs and Configuration Drive (config_drive).
- **Volume-backed live migration**. When instances are backed by volumes rather than ephemeral disk, no shared storage is required, and migration is supported (currently only in libvirt-based hypervisors).

The following sections describe how to configure your hosts and compute nodes for migrations by using the KVM and XenServer hypervisors.

KVM-Libvirt

Prerequisites

- **Hypervisor:** KVM with libvirt
- **Shared storage:** `NOVA-INST-DIR/instances/` (for example, `/var/lib/nova/instances`) has to be mounted by shared storage. This guide uses NFS but other options, including the OpenStack Gluster Connector are available.
- **Instances:** Instance can be migrated with iSCSI based volumes

Note

- Because the Compute service does not use the libvirt live migration functionality by default, guests are suspended before migration and might experience several minutes of downtime. For details, see the section called “Enable true live migration” [94].
- This guide assumes the default value for `instances_path` in your `nova.conf` file (`NOVA-INST-DIR/instances`). If you have changed the `state_path` or `instances_path` variables, modify accordingly.
- You must specify `vncserver_listen=0.0.0.0` or live migration does not work correctly.

Example Compute installation environment

- Prepare at least three servers; for example, `HostA`, `HostB`, and `HostC`:
  - `HostA` is the *Cloud Controller*, and should run these services: `nova-api`, `nova-scheduler`, `nova-network`, `cinder-volume`, and `nova-objectstore`.
  - `HostB` and `HostC` are the *compute nodes* that run `nova-compute`.

  Ensure that *`NOVA-INST-DIR`* (set with `state_path` in the `nova.conf` file) is the same on all hosts.
- In this example, `HostA` is the NFSv4 server that exports `NOVA-INST-DIR/instances`, and `HostB` and `HostC` mount it.

Procedure 4.4. To configure your system

1. Configure your DNS or `/etc/hosts` and ensure it is consistent across all hosts. Make sure that the three hosts can perform name resolution with each other. As a test, use the **ping** command to ping each host from one another.

   ```
   $ ping HostA
   $ ping HostB
   $ ping HostC
   ```

2. Ensure that the UID and GID of your Compute and libvirt users are identical between each of your servers. This ensures that the permissions on the NFS mount works correctly.

3. Export *`NOVA-INST-DIR`*`/instances` from `HostA`, and have it readable and writable by the Compute user on `HostB` and `HostC`.

   For more information, see: SettingUpNFSHowTo or CentOS / Redhat: Setup NFS v4.0 File Server

4. Configure the NFS server at `HostA` by adding the following line to the `/etc/exports` file:

   ```
   NOVA-INST-DIR/instances HostA/255.255.0.0(rw,sync,fsid=0,no_root_squash)
   ```

   Change the subnet mask (`255.255.0.0`) to the appropriate value to include the IP addresses of `HostB` and `HostC`. Then restart the NFS server:

   ```
   # /etc/init.d/nfs-kernel-server restart
   # /etc/init.d/idmapd restart
   ```

5. Set the 'execute/search' bit on your shared directory.

   On both compute nodes, make sure to enable the 'execute/search' bit to allow qemu to be able to use the images within the directories. On all hosts, run the following command:

   ```
   $ chmod o+x NOVA-INST-DIR/instances
   ```

6. Configure NFS at HostB and HostC by adding the following line to the `/etc/fstab` file:

   ```
   HostA:/ /NOVA-INST-DIR/instances nfs4 defaults 0 0
   ```

   Ensure that you can mount the exported directory can be mounted:

   ```
   $ mount -a -v
   ```

   Check that HostA can see the "*`NOVA-INST-DIR`*`/instances/`" directory:

   ```
   $ ls -ld NOVA-INST-DIR/instances/
   ```

   ```
   drwxr-xr-x 2 nova nova 4096 2012-05-19 14:34 nova-install-dir/instances/
   ```

   Perform the same check at HostB and HostC, paying special attention to the permissions (Compute should be able to write):

   ```
   $ ls -ld NOVA-INST-DIR/instances/
   ```

```
drwxr-xr-x 2 nova nova 4096 2012-05-07 14:34 nova-install-dir/instances/
```

```
$ df -k
```

```
Filesystem           1K-blocks      Used Available Use% Mounted on
/dev/sda1            921514972   4180880 870523828   1% /
none                  16498340      1228  16497112   1% /dev
none                  16502856         0  16502856   0% /dev/shm
none                  16502856       368  16502488   1% /var/run
none                  16502856         0  16502856   0% /var/lock
none                  16502856         0  16502856   0% /lib/init/rw
HostA:        921515008 101921792 772783104  12% /var/lib/nova/instances
 ( <--- this line is important.)
```

7. Update the libvirt configurations so that the calls can be made securely. These methods enable remote access over TCP and are not documented here, please consult your network administrator for assistance in deciding how to configure access.

   - SSH tunnel to libvirtd's UNIX socket
   - libvirtd TCP socket, with GSSAPI/Kerberos for auth+data encryption
   - libvirtd TCP socket, with TLS for encryption and x509 client certs for authentication
   - libvirtd TCP socket, with TLS for encryption and Kerberos for authentication

   Restart libvirt. After you run the command, ensure that libvirt is successfully restarted:

```
# stop libvirt-bin && start libvirt-bin
$ ps -ef | grep libvirt
```

```
root 1145 1 0 Nov27 ? 00:00:03 /usr/sbin/libvirtd -d -l
```

8. Configure your firewall to allow libvirt to communicate between nodes.

   By default, libvirt listens on TCP port 16509, and an ephemeral TCP range from 49152 to 49261 is used for the KVM communications. Based on the secure remote access TCP configuration you chose, be careful choosing what ports you open and understand who has access. For information about ports that are used with libvirt, see the libvirt documentation.

9. You can now configure options for live migration. In most cases, you do not need to configure any options. The following chart is for advanced usage only.

Table 4.6. Description of configuration options for livemigration

Configuration option = Default value	Description
[DEFAULT]	
live_migration_retry_count = 30	(IntOpt) Number of 1 second retries needed in live_migration
[libvirt]	
live_migration_bandwidth = 0	(IntOpt) Maximum bandwidth to be used during migration, in Mbps
live_migration_flag = VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER	(StrOpt) Migration flags to be set for live migration

Configuration option = Default value	Description
live_migration_uri = qemu+tcp://%s/system	(StrOpt) Migration target URI (any included "%s" is replaced with the migration target hostname)

Enable true live migration

By default, the Compute service does not use the libvirt live migration functionality. To enable this functionality, add the following line to the `nova.conf` file:

```
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,
VIR_MIGRATE_LIVE
```

The Compute service does not use libvirt's live migration by default because there is a risk that the migration process never ends. This can happen if the guest operating system dirties blocks on the disk faster than they can migrated.

XenServer

Shared storage

Prerequisites

- **Compatible XenServer hypervisors**. For more information, see the Requirements for Creating Resource Pools section of the *XenServer Administrator's Guide*.
- **Shared storage**. An NFS export, visible to all XenServer hosts.

Note

For the supported NFS versions, see the NFS VHD section of the *XenServer Administrator's Guide*.

To use shared storage live migration with XenServer hypervisors, the hosts must be joined to a XenServer pool. To create that pool, a host aggregate must be created with special metadata. This metadata is used by the XAPI plug-ins to establish the pool.

Procedure 4.5. To use shared storage live migration with XenServer hypervisors

1. Add an NFS VHD storage to your master XenServer, and set it as default SR. For more information, please refer to the NFS VHD section in the *XenServer Administrator's Guide*.

2. Configure all the compute nodes to use the default sr for pool operations. Add this line to your `nova.conf` configuration files across your compute nodes:

   ```
   sr_matching_filter=default-sr:true
   ```

3. Create a host aggregate:

   ```
   $ nova aggregate-create <name-for-pool> <availability-zone>
   ```

   The command displays a table that contains the ID of the newly created aggregate.

   Now add special metadata to the aggregate, to mark it as a hypervisor pool:

```
$ nova aggregate-set-metadata <aggregate-id> hypervisor_pool=true
```

```
$ nova aggregate-set-metadata <aggregate-id> operational_state=created
```

Make the first compute node part of that aggregate:

```
$ nova aggregate-add-host <aggregate-id> <name-of-master-compute>
```

At this point, the host is part of a XenServer pool.

4. Add additional hosts to the pool:

```
$ nova aggregate-add-host <aggregate-id> <compute-host-name>
```

Note

At this point, the added compute node and the host are shut down, to join the host to the XenServer pool. The operation fails, if any server other than the compute node is running/suspended on your host.

Block migration

Prerequisites

- **Compatible XenServer hypervisors**. The hypervisors must support the Storage XenMotion feature. See your XenServer manual to make sure your edition has this feature.

Note

- To use block migration, you must use the CHANGE THIS == ==*block-migrate* parameter with the live migration command.
- Block migration works only with EXT local storage SRs, and the server must not have any volumes attached.

Migrate instances

Before starting migrations, review the Configure migrations section.

Migration provides a scheme to migrate running instances from one OpenStack Compute server to another OpenStack Compute server.

Procedure 4.6. To migrate instances

1. Look at the running instances, to get the ID of the instance you wish to migrate.

```
$ nova list
+--------------------------------------+------+--------+-----------------+
|                  ID                  | Name | Status |Networks         |
+--------------------------------------+------+--------+-----------------+
| d1df1b5a-70c4-4fed-98b7-423362f2c47c | vm1  | ACTIVE | private=a.b.c.d |
| d693db9e-a7cf-45ef-a7c9-b3ecb5f22645 | vm2  | ACTIVE | private=e.f.g.h |
+--------------------------------------+------+--------+-----------------+
```

2. Look at information associated with that instance. This example uses 'vm1' from above.

```
$ nova show d1df1b5a-70c4-4fed-98b7-423362f2c47c
+-------------------------------------
+----------------------------------------------------------+
|               Property              |                          Value
                       |
+-------------------------------------
+----------------------------------------------------------+
...
| OS-EXT-SRV-ATTR:host                | HostB
                       |
...
| flavor                              | m1.tiny
                       |
| id                                  |
 d1df1b5a-70c4-4fed-98b7-423362f2c47c                   |
| name                                | vm1
                       |
| private network                     | a.b.c.d
                       |
| status                              | ACTIVE
                       |
...
+-------------------------------------
+----------------------------------------------------------+
```

In this example, vm1 is running on HostB.

3. Select the server to which instances will be migrated:

```
# nova service-list
+------------------+------------+----------+---------+-------
+----------------------------+-----------------+
| Binary           | Host       | Zone     | Status  | State | Updated_at
                | Disabled Reason |
+------------------+------------+----------+---------+-------
+----------------------------+-----------------+
| nova-consoleauth | HostA      | internal | enabled | up    |
 2014-03-25T10:33:25.000000 | -               |
| nova-scheduler   | HostA      | internal | enabled | up    |
 2014-03-25T10:33:25.000000 | -               |
| nova-conductor   | HostA      | internal | enabled | up    |
 2014-03-25T10:33:27.000000 | -               |
| nova-compute     | HostB      | nova     | enabled | up    |
 2014-03-25T10:33:31.000000 | -               |
| nova-compute     | HostC      | nova     | enabled | up    |
 2014-03-25T10:33:31.000000 | -               |
| nova-cert        | HostA      | internal | enabled | up    |
 2014-03-25T10:33:31.000000 | -               |
+------------------+-----------------------+----------+---------+-------
+----------------------------+-----------------+
```

In this example, HostC can be picked up because `nova-compute` is running on it.

4. Ensure that HostC has enough resources for migration.

```
# nova host-describe HostC
+-----------+------------+-----+-----------+---------+
| HOST        | PROJECT     | cpu | memory_mb | disk_gb |
+-----------+------------+-----+-----------+---------+
| HostC       | (total)     | 16   | 32232     | 878      |
| HostC       | (used_now) | 13   | 21284     | 442      |
| HostC       | (used_max) | 13   | 21284     | 442      |
| HostC       | p1          | 13   | 21284     | 442      |
| HostC       | p2          | 13   | 21284     | 442      |
+-----------+------------+-----+-----------+---------+
```

- **cpu:**the number of cpu
- **memory_mb:**total amount of memory (in MB)
- **disk_gb:**total amount of space for NOVA-INST-DIR/instances (in GB)
- **1st line shows** total amount of resources for the physical server.
- **2nd line shows** currently used resources.
- **3rd line shows** maximum used resources.
- **4th line and under** shows the resource for each project.

5. Use the **nova live-migration** command to migrate the instances:

```
$ nova live-migration server host_name
```

Where `server` can be either the server's ID or name. For example:

```
$ nova live-migration d1df1b5a-70c4-4fed-98b7-423362f2c47c HostC
Migration of d1df1b5a-70c4-4fed-98b7-423362f2c47c initiated.
```

Ensure instances are migrated successfully with **nova list**. If instances are still running on HostB, check log files (src/dest `nova-compute` and `nova-scheduler`) to determine why.

Note

Although the **nova** command is called **live-migration**, under the default Compute configuration options the instances are suspended before migration.

For more details, see Configure migrations in *OpenStack Configuration Reference*.

Configure remote console access

To provide a remote console or remote desktop access to guest virtual machines, use VNC or SPICE HTML5 through either the OpenStack dashboard or the command line. Best practice is to select one or the other to run.

VNC console proxy

The VNC proxy is an OpenStack component that enables compute service users to access their instances through VNC clients.

The VNC console connection works as follows:

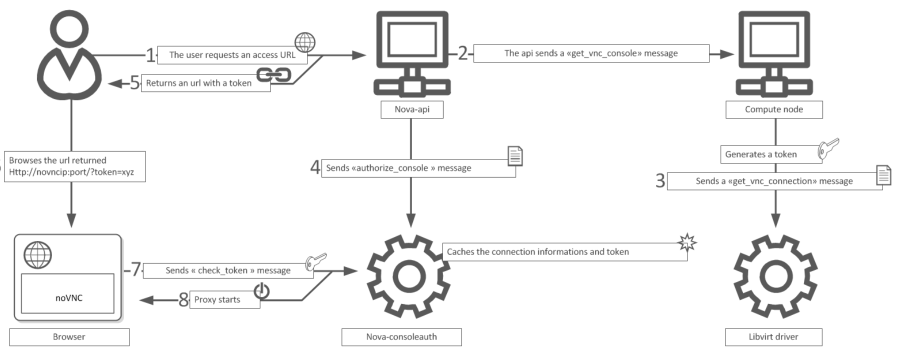
1. A user connects to the API and gets an `access_url` such as, `http://ip:port/?token=xyz`.
2. The user pastes the URL in a browser or uses it as a client parameter.
3. The browser or client connects to the proxy.
4. The proxy talks to `nova-consoleauth` to authorize the token for the user, and maps the token to the *private* host and port of the VNC server for an instance.

   The compute host specifies the address that the proxy should use to connect through the `nova.conf` file option, `vncserver_proxyclient_address`. In this way, the VNC proxy works as a bridge between the public network and private host network.
5. The proxy initiates the connection to VNC server and continues to proxy until the session ends.

The proxy also tunnels the VNC protocol over WebSockets so that the `noVNC` client can talk to VNC servers. In general, the VNC proxy:

- Bridges between the public network where the clients live and the private network where VNC servers live.
- Mediates token authentication.
- Transparently deals with hypervisor-specific connection details to provide a uniform client experience.

Figure 4.7. noVNC process


1 The user requests an access URL
2 The api sends a «get_vnc_console» message
Nova-api
Compute node
5 Returns an url with a token
6 Browses the url returned Http://novncip:port/?token=xyz
4 Sends «authorize_console » message
Generates a token
3 Sends a «get_vnc_connection» message
Caches the connection informations and token
7 Sends « check_token » message
8 Proxy starts
noVNC
Browser
Nova-consoleauth
Libvirt driver

About nova-consoleauth

Both client proxies leverage a shared service to manage token authentication called `nova-consoleauth`. This service must be running for either proxy to work. Many proxies of either type can be run against a single `nova-consoleauth` service in a cluster configuration.

Do not confuse the `nova-consoleauth` shared service with `nova-console`, which is a XenAPI-specific service that most recent VNC proxy architectures do not use.

Typical deployment

A typical deployment has the following components:

- A `nova-consoleauth` process. Typically runs on the controller host.
- One or more `nova-novncproxy` services. Supports browser-based noVNC clients. For simple deployments, this service typically runs on the same machine as `nova-api` because it operates as a proxy between the public network and the private compute host network.
- One or more `nova-xvpvncproxy` services. Supports the special Java client discussed here. For simple deployments, this service typically runs on the same machine as `nova-api` because it acts as a proxy between the public network and the private compute host network.
- One or more compute hosts. These compute hosts must have correctly configured options, as follows.

VNC configuration options

To customize the VNC console, use the following configuration options:

Table 4.7. Description of configuration options for vnc

Configuration option = Default value	**Description**
[DEFAULT]	
novncproxy_base_url = http://127.0.0.1:6080/vnc_auto.html	(StrOpt) Location of VNC console proxy, in the form "http://127.0.0.1:6080/vnc_auto.html"
vnc_enabled = True	(BoolOpt) Enable VNC related features
vnc_keymap = en-us	(StrOpt) Keymap for VNC
vncserver_listen = 127.0.0.1	(StrOpt) IP address on which instance vncservers should listen
vncserver_proxyclient_address = 127.0.0.1	(StrOpt) The address to which proxy clients (like nova-xvpvncproxy) should connect
[vmware]	
vnc_port = 5900	(IntOpt) VNC starting port
vnc_port_total = 10000	(IntOpt) Total number of VNC ports

Note

To support live migration, you cannot specify a specific IP address for `vncserver_listen`, because that IP address does not exist on the destination host.

Note

- The `vncserver_proxyclient_address` defaults to `127.0.0.1`, which is the address of the compute host that Compute instructs proxies to use when connecting to instance servers.
- For all-in-one XenServer domU deployments, set this to 169.254.0.1.

- For multi-host XenServer domU deployments, set to a dom0 management IP on the same network as the proxies.
- For multi-host libvirt deployments, set to a host management IP on the same network as the proxies.

nova-novncproxy (noVNC)

You must install the noVNC package, which contains the `nova-novncproxy` service. As root, run the following command:

```
# apt-get install novnc
```

The service starts automatically on installation.

To restart the service, run:

```
# service novnc restart
```

The configuration option parameter should point to your `nova.conf` file, which includes the message queue server address and credentials.

By default, `nova-novncproxy` binds on `0.0.0.0:6080`.

To connect the service to your Compute deployment, add the following configuration options to your `nova.conf` file:

- `vncserver_listen=`*`0.0.0.0`*

  Specifies the address on which the VNC service should bind. Make sure it is assigned one of the compute node interfaces. This address is the one used by your domain file.

  ```
   <graphics type="vnc" autoport="yes" keymap="en-us" listen="0.0.0.0"/>
  ```

  ### Note

  To use live migration, use the *`0.0.0.0`* address.

- `vncserver_proxyclient_address=`*`127.0.0.1`*

  The address of the compute host that Compute instructs proxies to use when connecting to instance `vncservers`.

Frequently asked questions about VNC access to virtual machines

- **Q: What is the difference between `nova-xvpvncproxy` and `nova-novncproxy`?**

  A: `nova-xvpvncproxy`, which ships with OpenStack Compute, is a proxy that supports a simple Java client. `nova-novncproxy` uses noVNC to provide VNC support through a web browser.

- **Q: I want VNC support in the OpenStack dashboard. What services do I need?**

A: You need `nova-novncproxy`, `nova-consoleauth`, and correctly configured compute hosts.

- **Q: When I use nova get-vnc-console or click on the VNC tab of the OpenStack dashboard, it hangs. Why?**

  A: Make sure you are running `nova-consoleauth` (in addition to `nova-novncproxy`). The proxies rely on `nova-consoleauth` to validate tokens, and waits for a reply from them until a timeout is reached.

- **Q: My VNC proxy worked fine during my all-in-one test, but now it doesn't work on multi host. Why?**

  A: The default options work for an all-in-one install, but changes must be made on your compute hosts once you start to build a cluster. As an example, suppose you have two servers:

```
PROXYSERVER (public_ip=172.24.1.1, management_ip=192.168.1.1)
COMPUTESERVER (management_ip=192.168.1.2)
```

  Your `nova-compute` configuration file must set the following values:

```
# These flags help construct a connection data structure
vncserver_proxyclient_address=192.168.1.2
novncproxy_base_url=http://172.24.1.1:6080/vnc_auto.html
xvpvncproxy_base_url=http://172.24.1.1:6081/console

# This is the address where the underlying vncserver (not the proxy)
# will listen for connections.
vncserver_listen=192.168.1.2
```

Note

`novncproxy_base_url` and `xvpvncproxy_base_url` use a public IP; this is the URL that is ultimately returned to clients, which generally do not have access to your private network. Your PROXYSERVER must be able to reach `vncserver_proxyclient_address`, because that is the address over which the VNC connection is proxied.

- **Q: My noVNC does not work with recent versions of web browsers. Why?**

  A: Make sure you have installed `python-numpy`, which is required to support a newer version of the WebSocket protocol (HyBi-07+).

- **Q: How do I adjust the dimensions of the VNC window image in the OpenStack dashboard?**

  A: These values are hard-coded in a Django HTML template. To alter them, edit the `_detail_vnc.html` template file. The location of this file varies based on Linux distribution. On Ubuntu 12.04, the file is at `/usr/share/pyshared/horizon/dashboards/nova/instances/templates/instances/_detail_vnc.html`.

  Modify the `width` and `height` options, as follows:

```
<iframe src="{{ vnc_url }}" width="720" height="430"></iframe>
```

SPICE console

OpenStack Compute supports VNC consoles to guests. The VNC protocol is fairly limited, lacking support for multiple monitors, bi-directional audio, reliable cut-and-paste, video streaming and more. SPICE is a new protocol that aims to address the limitations in VNC and provide good remote desktop support.

SPICE support in OpenStack Compute shares a similar architecture to the VNC implementation. The OpenStack dashboard uses a SPICE-HTML5 widget in its console tab that communicates to the `nova-spicehtml5proxy` service by using SPICE-over-websockets. The `nova-spicehtml5proxy` service communicates directly with the hypervisor process by using SPICE.

VNC must be explicitly disabled to get access to the SPICE console. Set the `vnc_enabled` option to `False` in the `[DEFAULT]` section to disable the VNC console.

Use the following options to configure SPICE as the console for OpenStack Compute:

Table 4.8. Description of configuration options for spice

Configuration option = Default value	Description
[spice]	
agent_enabled = True	(BoolOpt) Enable spice guest agent support
enabled = False	(BoolOpt) Enable spice related features
html5proxy_base_url = http://127.0.0.1:6082/spice_auto.html	(StrOpt) Location of spice HTML5 console proxy, in the form "http://127.0.0.1:6082/spice_auto.html"
keymap = en-us	(StrOpt) Keymap for spice
server_listen = 127.0.0.1	(StrOpt) IP address on which instance spice server should listen
server_proxyclient_address = 127.0.0.1	(StrOpt) The address to which proxy clients (like nova-spicehtml5proxy) should connect

Configure Compute service groups

To effectively manage and utilize compute nodes, the Compute service must know their statuses. For example, when a user launches a new VM, the Compute scheduler sends the request to a live node; the Compute service queries the ServiceGroup API to get information about whether a node is alive.

When a compute worker (running the `nova-compute` daemon) starts, it calls the `join` API to join the compute group. Any interested service (for example, the scheduler) can query the group's membership and the status of its nodes. Internally, the `ServiceGroup` client driver automatically updates the compute worker status.

The database, ZooKeeper, and Memcache drivers are available.

Database ServiceGroup driver

By default, Compute uses the database driver to track node liveness. In a compute worker, this driver periodically sends a **db update** command to the database, saying “I'm OK” with a timestamp. Compute uses a pre-defined timeout (`service_down_time`) to determine whether a node is dead.

The driver has limitations, which can be an issue depending on your setup. The more compute worker nodes that you have, the more pressure you put on the database. By default, the timeout is 60 seconds so it might take some time to detect node failures. You could reduce the timeout value, but you must also make the database update more frequently, which again increases the database workload.

The database contains data that is both transient (whether the node is alive) and persistent (for example, entries for VM owners). With the ServiceGroup abstraction, Compute can treat each type separately.

ZooKeeper ServiceGroup driver

The ZooKeeper ServiceGroup driver works by using ZooKeeper ephemeral nodes. ZooKeeper, in contrast to databases, is a distributed system. Its load is divided among several servers. At a compute worker node, after establishing a ZooKeeper session, the driver creates an ephemeral znode in the group directory. Ephemeral znodes have the same lifespan as the session. If the worker node or the `nova-compute` daemon crashes, or a network partition is in place between the worker and the ZooKeeper server quorums, the ephemeral znodes are removed automatically. The driver gets the group membership by running the **ls** command in the group directory.

To use the ZooKeeper driver, you must install ZooKeeper servers and client libraries. Setting up ZooKeeper servers is outside the scope of this guide (for more information, see Apache Zookeeper).

To use ZooKeeper, you must install client-side Python libraries on every nova node: `python-zookeeper` – the official Zookeeper Python binding and `evzookeeper` – the library to make the binding work with the eventlet threading model.

The following example assumes the ZooKeeper server addresses and ports are `192.168.2.1:2181`, `192.168.2.2:2181`, and `192.168.2.3:2181`.

The following values in the `/etc/nova/nova.conf` file (on every node) are required for the `ZooKeeper` driver:

```
# Driver for the ServiceGroup serice
servicegroup_driver="zk"

[zookeeper]
address="192.168.2.1:2181,192.168.2.2:2181,192.168.2.3:2181"
```

To customize the Compute Service groups, use the following configuration option settings:

Table 4.9. Description of configuration options for zookeeper

Configuration option = Default value	Description
[zookeeper]	
address = None	(StrOpt) The ZooKeeper addresses for servicegroup service in the format of host1:port,host2:port,host3:port
recv_timeout = 4000	(IntOpt) The recv_timeout parameter for the zk session
sg_prefix = /servicegroups	(StrOpt) The prefix used in ZooKeeper to store ephemeral nodes
sg_retry_interval = 5	(IntOpt) Number of seconds to wait until retrying to join the session

Memcache ServiceGroup driver

The `memcache` ServiceGroup driver uses memcached, which is a distributed memory object caching system that is often used to increase site performance. For more details, see memcached.org.

To use the `memcache` driver, you must install `memcached`. However, because `memcached` is often used for both OpenStack Object Storage and OpenStack dashboard, it might already be installed. If `memcached` is not installed, refer to the *OpenStack Installation Guide* for more information.

The following values in the `/etc/nova/nova.conf` file (on every node) are required for the `memcache` driver:

```
# Driver for the ServiceGroup serice
servicegroup_driver="mc"

# Memcached servers. Use either a list of memcached servers to use for caching
 (list value),
# or "<None>" for in-process caching (default).
memcached_servers=<None>

# Timeout; maximum time since last check-in for up service (integer value).
# Helps to define whether a node is dead
service_down_time=60
```

Security hardening

OpenStack Compute can be integrated with various third-party technologies to increase security. For more information, see the *OpenStack Security Guide*.

Trusted compute pools

Trusted compute pools enable administrators to designate a group of compute hosts as trusted. These hosts use hardware-based security features, such as the Intel Trusted Execution Technology (TXT), to provide an additional level of security. Combined with an external stand-alone, web-based remote attestation server, cloud providers can ensure that the compute node runs only software with verified measurements and can ensure a secure cloud stack.

Using the trusted compute pools, cloud subscribers can request services to run on verified compute nodes.

The remote attestation server performs node verification as follows:

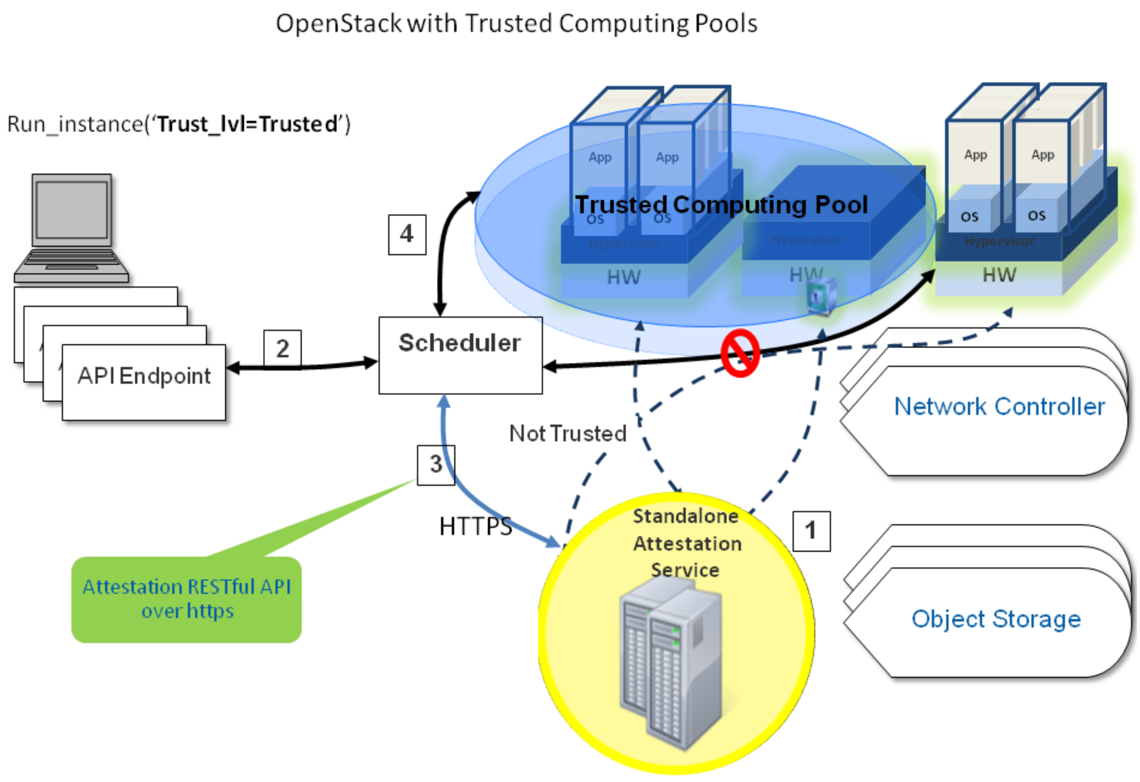
1. Compute nodes boot with Intel TXT technology enabled.

2. The compute node BIOS, hypervisor, and OS are measured.

3. Measured data is sent to the attestation server when challenged by the attestation server.

4. The attestation server verifies those measurements against a good and known database to determine node trustworthiness.

A description of how to set up an attestation service is beyond the scope of this document. For an open source project that you can use to implement an attestation service, see the Open Attestation project.


OpenStack with Trusted Computing Pools
Run_instance('Trust_lvl=Trusted')
API Endpoint
Scheduler
Trusted Computing Pool
App
OS
Hypervisor
HW
Not Trusted
HTTPS
Standalone Attestation Service
Attestation RESTful API over https
Network Controller
Object Storage
1
2
3
4

Configure Compute to use trusted compute pools

1. Enable scheduling support for trusted compute pools by adding the following lines in the DEFAULT section in the `/etc/nova/nova.conf` file:

```
[DEFAULT]
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter,
TrustedFilter
```

2. Specify the connection information for your attestation service by adding the following lines to the `trusted_computing` section in the `/etc/nova/nova.conf` file:

```
[trusted_computing]
server=10.1.71.206
port=8443
server_ca_file=/etc/nova/ssl.10.1.71.206.crt
# If using OAT v1.5, use this api_url:
api_url=/AttestationService/resources
# If using OAT pre-v1.5, use this api_url:
#api_url=/OpenAttestationWebServices/V1.0
auth_blob=i-am-openstack
```

Where:

server	Host name or IP address of the host that runs the attestation service.
port	HTTPS port for the attestation service.
server_ca_file	Certificate file used to verify the attestation server's identity.
api_url	The attestation service's URL path.
auth_blob	An authentication blob, which is required by the attestation service.

3. Restart the `nova-compute` and `nova-scheduler` services.

Configuration reference

To customize the trusted compute pools, use the following configuration option settings:

Table 4.10. Description of configuration options for trustedcomputing

Configuration option = Default value	Description
[trusted_computing]	
attestation_api_url = /OpenAttestationWebServices/V1.0	(StrOpt) Attestation web API URL
attestation_auth_blob = None	(StrOpt) Attestation authorization blob - must change
attestation_auth_timeout = 60	(IntOpt) Attestation status cache valid period length
attestation_port = 8443	(StrOpt) Attestation server port
attestation_server = None	(StrOpt) Attestation server HTTP
attestation_server_ca_file = None	(StrOpt) Attestation server Cert file for Identity verification

Specify trusted flavors

To designate hosts as trusted:

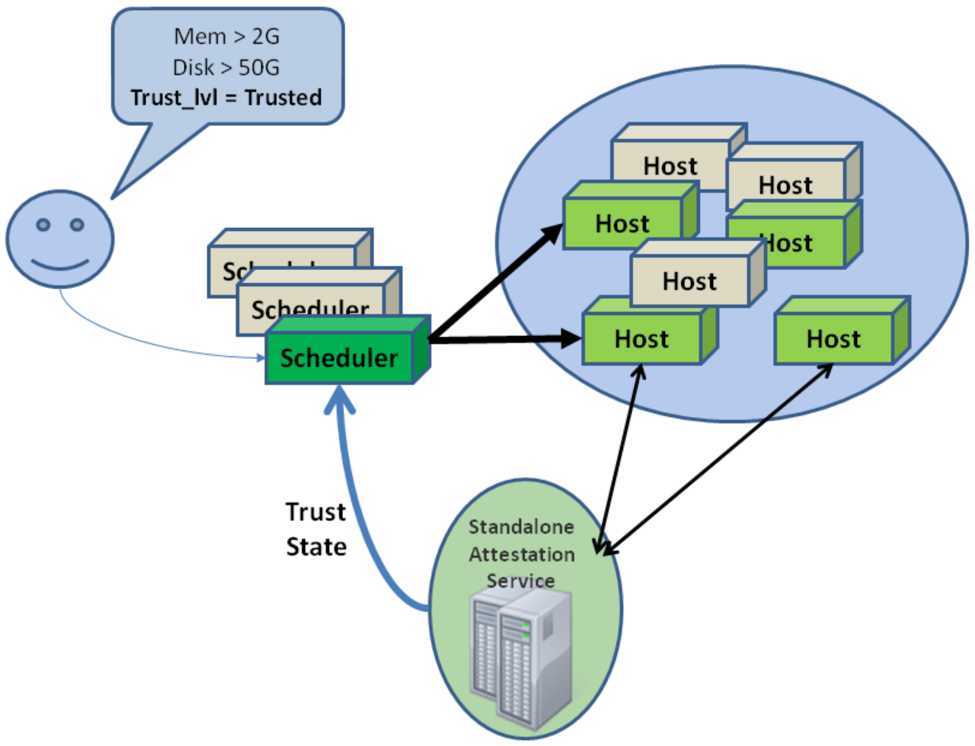
1. Configure one or more flavors as trusted by using the **nova flavor-key set** command. For example, to set the `m1.tiny` flavor as trusted:

   ```
   $ nova flavor-key m1.tiny set trust:trusted_host trusted
   ```

2. Request that your instance be run on a trusted host, by specifying a trusted flavor when booting the instance. For example:

   ```
   $ nova boot --flavor m1.tiny --key_name myKeypairName --image myImageID
    newInstanceName
   ```

Figure 4.8. Trusted compute pool



Encrypt Compute metadata traffic

OpenStack Juno supports encrypting Compute metadata traffic with HTTPS. Enable SSL encryption in the `metadata_agent.ini` file:

1. Enable the HTTPS protocol:

```
nova_metadata_protocol = https
```

2. Determine whether insecure SSL connections are accepted for Compute metadata server requests. The default value is `False`:

```
nova_metadata_insecure = False
```

3. Specify the path to the client certificate:

```
nova_client_cert = PATH_TO_CERT
```

4. Specify the path to the private key:

```
nova_client_priv_key = PATH_TO_KEY
```

Recover from a failed compute node

If you deployed Compute with a shared file system, you can quickly recover from a failed compute node. Of the two methods covered in these sections, evacuating is the preferred method even in the absence of shared storage. Evacuating provides many benefits over manual recovery, such as re-attachment of volumes and floating IPs.

Evacuate instances

If a cloud compute node fails due to a hardware malfunction or another reason, you can evacuate instances to make them available again. You can choose evacuation parameters for your use case.

To preserve user data on server disk, you must configure shared storage on the target host. Also, you must validate that the current VM host is down; otherwise, the evacuation fails with an error.

1. To list hosts and find a different host for the evacuated instance, run:

   ```
   $ nova host-list
   ```

2. Evacuate the instance. You can pass the instance password to the command by using the `--password <pwd>` option. If you do not specify a password, one is generated and printed after the command finishes successfully. The following command evacuates a server without shared storage from a host that is down to the specified *`host_b`*:

   ```
   $ nova evacuate evacuated_server_name host_b
   ```

   The instance is booted from a new disk, but preserves its configuration including its ID, name, uid, IP address, and so on. The command returns a password:

   ```
   +-----------+--------------+
   | Property  |    Value     |
   +-----------+--------------+
   | adminPass | kRAJpErnT4xZ |
   +-----------+--------------+
   ```

3. To preserve the user disk data on the evacuated server, deploy OpenStack Compute with a shared file system. To configure your system, see Configure migrations in *OpenStack Configuration Reference*. In the following example, the password remains unchanged:

   ```
   $ nova evacuate evacuated_server_name host_b --on-shared-storage
   ```

Manual recovery

To recover a KVM/libvirt compute node, see the previous section. Use the following procedure for all other hypervisors.

Procedure 4.7. Review host information

1. Identify the VMs on the affected hosts, using tools such as a combination of `nova list` and `nova show` or `euca-describe-instances`. For example, the following

output displays information about instance `i-000015b9` that is running on node `np-rcc54`:

```
$ euca-describe-instances
i-000015b9 at3-ui02 running nectarkey (376, np-rcc54) 0 m1.xxlarge
 2012-06-19T00:48:11.000Z 115.146.93.60
```

2. Review the status of the host by querying the Compute database. Some of the important information is highlighted below. The following example converts an EC2 API instance ID into an OpenStack ID; if you used the `nova` commands, you can substitute the ID directly. You can find the credentials for your database in `/etc/nova.conf`.

```
mysql> SELECT * FROM instances WHERE id = CONV('15b9', 16, 10) \G;
*************************** 1. row ***************************
              created_at: 2012-06-19 00:48:11
              updated_at: 2012-07-03 00:35:11
              deleted_at: NULL
...
                      id: 5561
...
             power_state: 5
                vm_state: shutoff
...
                hostname: at3-ui02
                    host: np-rcc54
...
                    uuid: 3f57699a-e773-4650-a443-b4b37eed5a06
...
              task_state: NULL
...
```

Procedure 4.8. Recover the VM

1. After you have determined the status of the VM on the failed host, decide to which compute host the affected VM should be moved. For example, run the following database command to move the VM to `np-rcc46`:

```
mysql> UPDATE instances SET host = 'np-rcc46' WHERE uuid = '3f57699a-
e773-4650-a443-b4b37eed5a06';
```

2. If using a hypervisor that relies on libvirt (such as KVM), it is a good idea to update the `libvirt.xml` file (found in `/var/lib/nova/instances/[instance ID]`). The important changes to make are:

   - Change the `DHCPSERVER` value to the host IP address of the compute host that is now the VM's new home.
   - Update the VNC IP, if it isn't already updated, to: `0.0.0.0`.

3. Reboot the VM:

```
$ nova reboot --hard 3f57699a-e773-4650-a443-b4b37eed5a06
```

In theory, the above database update and `nova reboot` command are all that is required to recover a VM from a failed host. However, if further problems occur, consider looking at recreating the network filter configuration using `virsh`, restarting the Compute services or updating the `vm_state` and `power_state` in the Compute database.

Recover from a UID/GID mismatch

When running OpenStack Compute, using a shared file system or an automated configuration tool, you could encounter a situation where some files on your compute node are using the wrong UID or GID. This causes a number of errors, such as being unable to do live migration or start virtual machines.

The following procedure runs on `nova-compute` hosts, based on the KVM hypervisor, and could help to restore the situation:

Procedure 4.9. To recover from a UID/GID mismatch

1. Ensure you do not use numbers that are already used for some other user/group.
2. Set the nova uid in `/etc/passwd` to the same number in all hosts (for example, 112).
3. Set the libvirt-qemu uid in `/etc/passwd` to the same number in all hosts (for example, 119).
4. Set the nova group in `/etc/group` file to the same number in all hosts (for example, 120).
5. Set the libvirtd group in `/etc/group` file to the same number in all hosts (for example, 119).
6. Stop the services on the compute node.
7. Change all the files owned by user `nova` or by group `nova`. For example:

   ```
   # find / -uid 108 -exec chown nova {} \; # note the 108 here is the old
    nova uid before the change
   # find / -gid 120 -exec chgrp nova {} \;
   ```

8. Repeat the steps for the libvirt-qemu owned files if those needed to change.
9. Restart the services.
10. Now you can run the **find** command to verify that all files using the correct identifiers.

Recover cloud after disaster

Use the following procedures to manage your cloud after a disaster, and to easily back up its persistent storage volumes. Backups **are** mandatory, even outside of disaster scenarios.

For a DRP definition, see http://en.wikipedia.org/wiki/Disaster_Recovery_Plan.

Disaster recovery example

A disaster could happen to several components of your architecture (for example, a disk crash, a network loss, or a power cut). In this example, the following components are configured:

1. A cloud controller (`nova-api`, `nova-objectstore`, `nova-network`)

2. A compute node (`nova-compute`)

3. A Storage Area Network (SAN) used by OpenStack Block Storage (`cinder-volumes`)

The worst disaster for a cloud is a power loss, which applies to all three components. Before a power loss:

- From the SAN to the cloud controller, we have an active iSCSI session (used for the "cinder-volumes" LVM's VG).
- From the cloud controller to the compute node, we also have active iSCSI sessions (managed by `cinder-volume`).
- For every volume, an iSCSI session is made (so 14 ebs volumes equals 14 sessions).
- From the cloud controller to the compute node, we also have iptables/ ebtables rules which allow access from the cloud controller to the running instance.
- And at least, from the cloud controller to the compute node; saved into database, the current state of the instances (in that case "running"), and their volumes attachment (mount point, volume ID, volume status, and so on.)

After the power loss occurs and all hardware components restart:

- From the SAN to the cloud, the iSCSI session no longer exists.
- From the cloud controller to the compute node, the iSCSI sessions no longer exist.
- From the cloud controller to the compute node, the iptables and ebtables are recreated, since at boot, `nova-network` reapplies configurations.
- From the cloud controller, instances are in a shutdown state (because they are no longer running).
- In the database, data was not updated at all, since Compute could not have anticipated the crash.

Before going further, and to prevent the administrator from making fatal mistakes, **instances won't be lost**, because no "**destroy**" or "**terminate**" command was invoked, so the files for the instances remain on the compute node.

Perform these tasks in the following order.

Warning

Do not add any extra steps at this stage.

1. Get the current relation from a volume to its instance, so that you can recreate the attachment.

2. Update the database to clean the stalled state. (After that, you cannot perform the first step).

3. Restart the instances. In other words, go from a shutdown to running state.
4. After the restart, reattach the volumes to their respective instances (optional).
5. SSH into the instances to reboot them.

Recover after a disaster

Procedure 4.10. To perform disaster recovery

1. **Get the instance-to-volume relationship**

   You must determine the current relationship from a volume to its instance, because you will re-create the attachment.

   You can find this relationship by running **nova volume-list**. Note that the **nova** client includes the ability to get volume information from OpenStack Block Storage.

2. **Update the database**

   Update the database to clean the stalled state. You must restore for every volume, using these queries to clean up the database:

   ```
   mysql> use cinder;
   mysql> update volumes set mountpoint=NULL;
   mysql> update volumes set status="available" where status
    <>"error_deleting";
   mysql> update volumes set attach_status="detached";
   mysql> update volumes set instance_id=0;
   ```

   You can then run **nova volume-list** commands to list all volumes.

3. **Restart instances**

   Restart the instances using the **nova reboot *$instance*** command.

   At this stage, depending on your image, some instances completely reboot and become reachable, while others stop on the "plymouth" stage.

4. **DO NOT reboot a second time**

   Do not reboot instances that are stopped at this point. Instance state depends on whether you added an `/etc/fstab` entry for that volume. Images built with the cloud-init package remain in a pending state, while others skip the missing volume and start. The idea of that stage is only to ask Compute to reboot every instance, so the stored state is preserved. For more information about cloud-init, see help.ubuntu.com/community/CloudInit.

5. **Reattach volumes**

   After the restart, and Compute has restored the right status, you can reattach the volumes to their respective instances using the **nova volume-attach** command. The following snippet uses a file of listed volumes to reattach them:

```
#!/bin/bash

while read line; do
    volume=`echo $line | $CUT -f 1 -d " "`
    instance=`echo $line | $CUT -f 2 -d " "`
    mount_point=`echo $line | $CUT -f 3 -d " "`
        echo "ATTACHING VOLUME FOR INSTANCE - $instance"
    nova volume-attach $instance $volume $mount_point
    sleep 2
done < $volumes_tmp_file
```

At this stage, instances that were pending on the boot sequence (plymouth) automatically continue their boot, and restart normally, while the ones that booted see the volume.

6. **SSH into instances**

   If some services depend on the volume, or if a volume has an entry into `fstab`, you should now simply restart the instance. This restart needs to be made from the instance itself, not through **nova**.

   SSH into the instance and perform a reboot:

   ```
   # shutdown -r now
   ```

By completing this procedure, you can successfully recover your cloud.

Note

Follow these guidelines:

- Use the *`errors=remount`* parameter in the `fstab` file, which prevents data corruption.

  The system locks any write to the disk if it detects an I/O error. This configuration option should be added into the `cinder-volume` server (the one which performs the iSCSI connection to the SAN), but also into the instances' `fstab` file.

- Do not add the entry for the SAN's disks to the `cinder-volume`'s `fstab` file.

  Some systems hang on that step, which means you could lose access to your cloud-controller. To re-run the session manually, run the following command before performing the mount:

  ```
  # iscsiadm -m discovery -t st -p $SAN_IP $ iscsiadm -m node --
  target-name $IQN -p $SAN_IP -l
  ```

- For your instances, if you have the whole `/home/` directory on the disk, leave a user's directory with the user's bash files and the `authorized_keys` file (instead of emptying the `/home` directory and mapping the disk on it).

  This enables you to connect to the instance, even without the volume attached, if you allow only connections through public keys.

Script the DRP

You can download from here a bash script which performs the following steps:

1. An array is created for instances and their attached volumes.
2. The MySQL database is updated.
3. Using `euca2ools`, all instances are restarted.
4. The volume attachment is made.
5. An SSH connection is performed into every instance using Compute credentials.

The "test mode" allows you to perform that whole sequence for only one instance.

To reproduce the power loss, connect to the compute node which runs that same instance and close the iSCSI session. Do not detach the volume using the **nova volume-detach** command; instead, manually close the iSCSI session. For the following example command uses an iSCSI session with the number 15:

```
# iscsiadm -m session -u -r 15
```

Do not forget the `-r` flag. Otherwise, you close ALL sessions.

Troubleshoot Compute

Common problems for Compute typically involve misconfigured networking or credentials that are not sourced properly in the environment. Also, most flat networking configurations do not enable **ping** or **ssh** from a compute node to the instances that run on that node. Another common problem is trying to run 32-bit images on a 64-bit compute node. This section shows you how to troubleshoot Compute.

Compute service logging

Compute stores a log file for each service in `/var/log/nova`. For example, `nova-compute.log` is the log for the `nova-compute` service. You can set the following options to format log strings for the nova.log module in the `nova.conf` file:

- `logging_context_format_string`
- `logging_default_format_string`

If the log level is set to `debug`, you can also specify `logging_debug_format_suffix` to append extra formatting. For information about what variables are available for the formatter see: http://docs.python.org/library/logging.html#formatter.

You have two options for logging for OpenStack Compute based on configuration settings. In `nova.conf`, include the `logfile` option to enable logging. Alternatively you can set `use_syslog=1` so that the nova daemon logs to syslog.

Guru Meditation reports

A Guru Meditation report is sent by the Compute Service upon receipt of the `SIGUSR1` signal. This report is a general-purpose error report, including a complete report of the service's current state, and is sent to `stderr`.

For example, if you redirect error output to `nova-api-err.log` using **nova-api 2>/var/log/nova/nova-api-err.log**, resulting in the process ID 8675, you can then run:

```
# kill -USR1 8675
```

This command triggers the Guru Meditation report to be printed to `/var/log/nova/nova-api-err.log`.

The report has the following sections:

- Package — Displays information about the package to which the process belongs, including version information.
- Threads — Displays stack traces and thread IDs for each of the threads within the process.
- Green Threads — Displays stack traces for each of the green threads within the process (green threads do not have thread IDs).
- Configuration — Lists all configuration options currently accessible through the CONF object for the current process.

For more information, see Guru Meditation Reports.

Common errors and fixes for Compute

The ask.openstack.org site offers a place to ask and answer questions, and you can also mark questions as frequently asked questions. This section describes some errors people have posted previously. Bugs are constantly being fixed, so online resources are a great way to get the most up-to-date errors and fixes.

Credential errors, 401, and 403 forbidden errors

Missing credentials cause a 403 forbidden error. To resolve this issue, use one of these methods:

1. **Manual method**. Get get the `novarc` file from the project ZIP file, save existing credentials in case of override. and manually source the `novarc` file.
2. **Script method**. Generates `novarc` from the project ZIP file and sources it for you.

When you run `nova-api` the first time, it generates the certificate authority information, including `openssl.cnf`. If you start the CA services before this, you might not be able to create your ZIP file. Restart the services. When your CA information is available, create your ZIP file.

Also, check your HTTP proxy settings to see whether they cause problems with `novarc` creation.

Instance errors

Sometimes a particular instance shows `pending` or you cannot SSH to it. Sometimes the image itself is the problem. For example, when you use flat manager networking, you do not have a DHCP server and certain images do not support interface injection; you cannot connect to them. The fix for this problem is to use an image that does support this method, such as Ubuntu, which obtains an IP address correctly with FlatManager network settings.

To troubleshoot other possible problems with an instance, such as an instance that stays in a spawning state, check the directory for the particular instance under `/var/lib/nova/instances` on the `nova-compute` host and make sure that these files are present:

- `libvirt.xml`
- `disk`
- `disk-raw`
- `kernel`
- `ramdisk`
- After the instance starts, `console.log`

If any files are missing, empty, or very small, the `nova-compute` service did not successfully download the images from the Image Service.

Also check `nova-compute.log` for exceptions. Sometimes they do not appear in the console output.

Next, check the log file for the instance in the `/var/log/libvirt/qemu` directory to see if it exists and has any useful error messages in it.

Finally, from the `/var/lib/nova/instances` directory for the instance, see if this command returns an error:

```
# virsh create libvirt.xml
```

Empty log output for Linux instances

You can view the log output of running instances from either the **Log** tab of the dashboard or the output of **nova console-log**. In some cases, the log output of a running Linux instance will be empty or only display a single character (for example, the **?** character).

This occurs when the Compute service attempts to retrieve the log output of the instance via a serial console while the instance itself is not configured to send output to the console. To rectify this, append the following parameters to kernel arguments specified in the instance's boot loader:

```
console=tty0 console=ttyS0,115200n8
```

Upon rebooting, the instance will be configured to send output to the Compute service.

Reset the state of an instance

If an instance remains in an intermediate state, such as `deleting`, you can use the **nova reset-state** command to manually reset the state of an instance to an error state. You can then delete the instance. For example:

```
$ nova reset-state c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
$ nova delete c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

You can also use the `--active` parameter to force the instance back to an active state instead of an error state. For example:

```
$ nova reset-state --active c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

Injection problems

If instances do not boot or boot slowly, investigate file injection as a cause.

To disable injection in libvirt, set `libvirt_inject_partition` to `-2`.

Note

If you have not enabled the configuration drive and you want to make user-specified files available from the metadata server for to improve performance and avoid boot failure if injection fails, you must disable injection.

5. Object Storage

Table of Contents

Introduction to Object Storage

OpenStack Object Storage (code-named Swift) is open source software for creating redundant, scalable data storage using clusters of standardized servers to store petabytes of accessible data. It is a long-term storage system for large amounts of static data that can be retrieved, leveraged, and updated. Object Storage uses a distributed architecture with no central point of control, providing greater scalability, redundancy, and permanence. Objects are written to multiple hardware devices, with the OpenStack software responsible for ensuring data replication and integrity across the cluster. Storage clusters scale horizontally by adding new nodes. Should a node fail, OpenStack works to replicate its content from other active nodes. Because OpenStack uses software logic to ensure data replication and distribution across different devices, inexpensive commodity hard drives and servers can be used in lieu of more expensive equipment.

Object Storage is ideal for cost effective, scale-out storage. It provides a fully distributed, API-accessible storage platform that can be integrated directly into applications or used for backup, archiving, and data retention.

Features and benefits

Features	Benefits
Leverages commodity hardware	No lock-in, lower price/GB
HDD/node failure agnostic	Self-healing, reliable, data redundancy protects from failures
Unlimited storage	Large and flat namespace, highly scalable read/write access, able to serve content directly from storage system
Multi-dimensional scalability	Scale-out architecture: Scale vertically and horizontally-distributed storage Backs up and archives large amounts of data with linear performance
Account/container/object structure	No nesting, not a traditional file system: Optimized for scale, it scales to multiple petabytes and billions of objects
Built-in replication 3# + data redundancy (compared with 2# on RAID)	A configurable number of accounts, containers and object copies for high availability

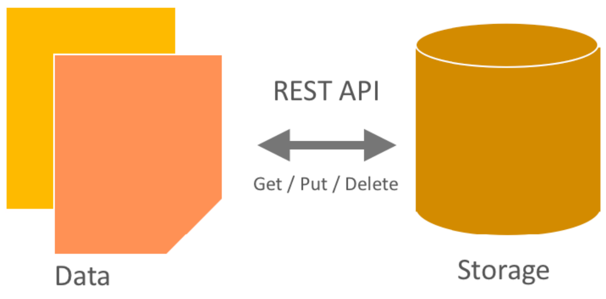
Easily add capacity (unlike RAID resize)	Elastic data scaling with ease
No central database	Higher performance, no bottlenecks
RAID not required	Handle many small, random reads and writes efficiently
Built-in management utilities	Account management: Create, add, verify, and delete users; Container management: Upload, download, and verify; Monitoring: Capacity, host, network, log trawling, and cluster health
Drive auditing	Detect drive failures preempting data corruption
Expiring objects	Users can set an expiration time or a TTL on an object to control access
Direct object access	Enable direct browser access to content, such as for a control panel
Realtime visibility into client requests	Know what users are requesting
Supports S3 API	Utilize tools that were designed for the popular S3 API
Restrict containers per account	Limit access to control usage by user
Support for NetApp, Nexenta, SolidFire	Unified support for block volumes using a variety of storage systems
Snapshot and backup API for block volumes	Data protection and recovery for VM data
Standalone volume API available	Separate endpoint and API for integration with other compute systems
Integration with Compute	Fully integrated with Compute for attaching block volumes and reporting on usage

Object Storage characteristics

The key characteristics of Object Storage are that:

- All objects stored in Object Storage have a URL.
- All objects stored are replicated 3# in as-unique-as-possible zones, which can be defined as a group of drives, a node, a rack, and so on.
- All objects have their own metadata.
- Developers interact with the object storage system through a RESTful HTTP API.
- Object data can be located anywhere in the cluster.
- The cluster scales by adding additional nodes without sacrificing performance, which allows a more cost-effective linear storage expansion than fork-lift upgrades.
- Data doesn't have to be migrate to an entirely new storage system.
- New nodes can be added to the cluster without downtime.
- Failed nodes and disks can be swapped out without downtime.
- It runs on industry-standard hardware, such as Dell, HP, and Supermicro.

Figure 5.1. Object Storage (Swift)



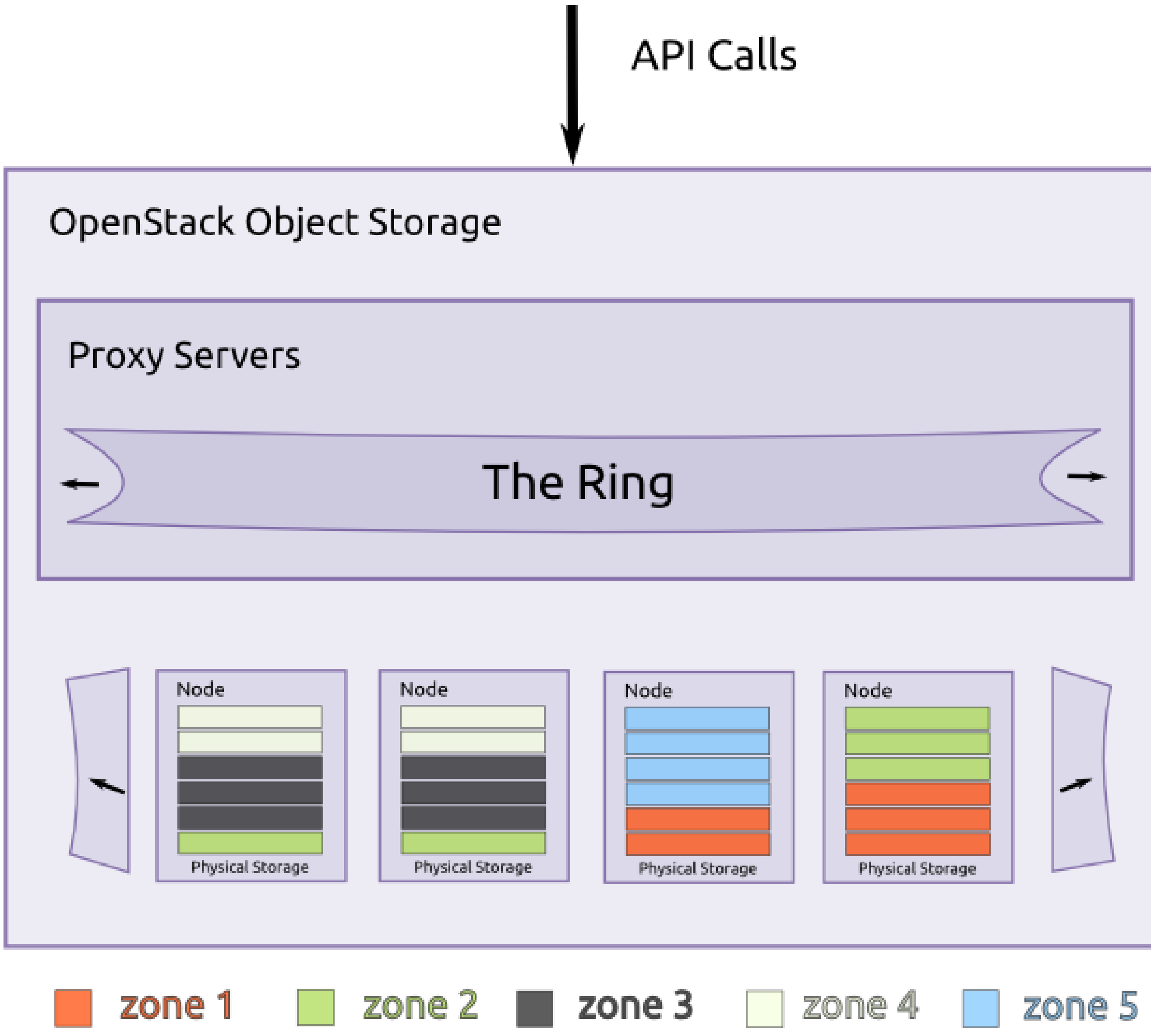
Developers can either write directly to the Swift API or use one of the many client libraries that exist for all of the popular programming languages, such as Java, Python, Ruby, and C#. Amazon S3 and RackSpace Cloud Files users should be very familiar with Object Storage. Users new to object storage systems will have to adjust to a different approach and mindset than those required for a traditional filesystem.

Components

The components that enable Object Storage to deliver high availability, high durability, and high concurrency are:

- **Proxy servers.** Handle all of the incoming API requests.
- **Rings.** Map logical names of data to locations on particular disks.
- **Zones.** Isolate data from other zones. A failure in one zone doesn't impact the rest of the cluster because data is replicated across zones.
- **Accounts and containers.** Each account and container are individual databases that are distributed across the cluster. An account database contains the list of containers in that account. A container database contains the list of objects in that container.
- **Objects.** The data itself.
- **Partitions.** A partition stores objects, account databases, and container databases and helps manage locations where data lives in the cluster.

Figure 5.2. Object Storage building blocks



Proxy servers

Proxy servers are the public face of Object Storage and handle all of the incoming API requests. Once a proxy server receives a request, it determines the storage node based on the object's URL, for example, https://swift.example.com/v1/account/container/object. Proxy servers also coordinate responses, handle failures, and coordinate timestamps.

Proxy servers use a shared-nothing architecture and can be scaled as needed based on projected workloads. A minimum of two proxy servers should be deployed for redundancy. If one proxy server fails, the others take over.

Rings

A ring represents a mapping between the names of entities stored on disk and their physical locations. There are separate rings for accounts, containers, and objects. When other components need to perform any operation on an object, container, or account, they need to interact with the appropriate ring to determine their location in the cluster.

The ring maintains this mapping using zones, devices, partitions, and replicas. Each partition in the ring is replicated, by default, three times across the cluster, and partition locations are stored in the mapping maintained by the ring. The ring is also responsible for determining which devices are used for handoff in failure scenarios.

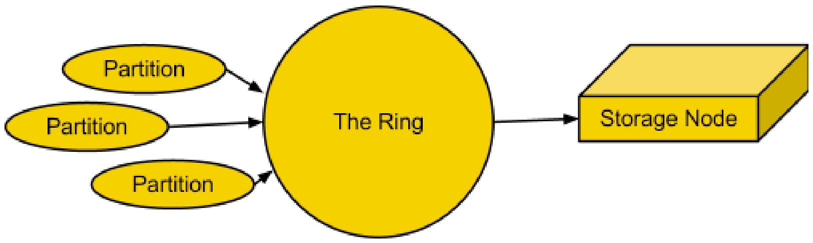
Data can be isolated into zones in the ring. Each partition replica is guaranteed to reside in a different zone. A zone could represent a drive, a server, a cabinet, a switch, or even a data center.

The partitions of the ring are equally divided among all of the devices in the Object Storage installation. When partitions need to be moved around (for example, if a device is added to the cluster), the ring ensures that a minimum number of partitions are moved at a time, and only one replica of a partition is moved at a time.

Weights can be used to balance the distribution of partitions on drives across the cluster. This can be useful, for example, when differently sized drives are used in a cluster.

The ring is used by the proxy server and several background processes (like replication).

Figure 5.3. The ring



These rings are externally managed, in that the server processes themselves do not modify the rings, they are instead given new rings modified by other tools.

The ring uses a configurable number of bits from a path’s MD5 hash as a partition index that designates a device. The number of bits kept from the hash is known as the partition power, and 2 to the partition power indicates the partition count. Partitioning the full MD5 hash ring allows other parts of the cluster to work in batches of items at once which ends up either more efficient or at least less complex than working with each item separately or the entire cluster all at once.

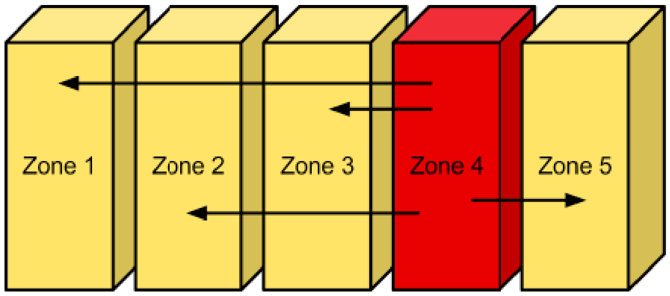
Another configurable value is the replica count, which indicates how many of the partition-device assignments make up a single ring. For a given partition number, each replica’s device will not be in the same zone as any other replica's device. Zones can be used to group devices based on physical locations, power separations, network separations, or any other attribute that would improve the availability of multiple replicas at the same time.

Zones

Object Storage allows configuring zones in order to isolate failure boundaries. Each data replica resides in a separate zone, if possible. At the smallest level, a zone could be a single drive or a grouping of a few drives. If there were five object storage servers, then each server would represent its own zone. Larger deployments would have an entire rack (or multiple racks) of object servers, each representing a zone. The goal of zones is to allow the cluster to tolerate significant outages of storage servers without losing all replicas of the data.

As mentioned earlier, everything in Object Storage is stored, by default, three times. Swift will place each replica "as-uniquely-as-possible" to ensure both high availability and high durability. This means that when chosing a replica location, Object Storage chooses a server in an unused zone before an unused server in a zone that already has a replica of the data.

Figure 5.4. Zones

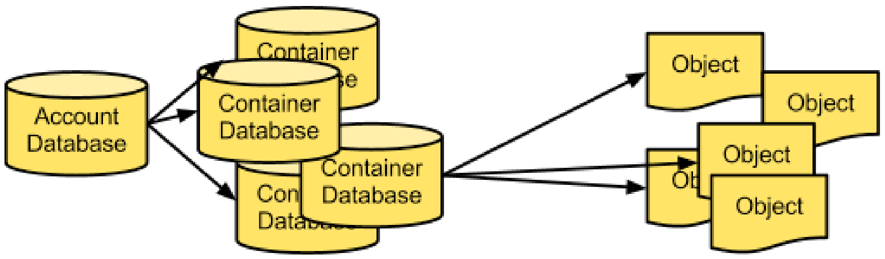


When a disk fails, replica data is automatically distributed to the other zones to ensure there are three copies of the data.

Accounts and containers

Each account and container is an individual SQLite database that is distributed across the cluster. An account database contains the list of containers in that account. A container database contains the list of objects in that container.

Figure 5.5. Accounts and containers



To keep track of object data locations, each account in the system has a database that references all of its containers, and each container database references each object.

Partitions

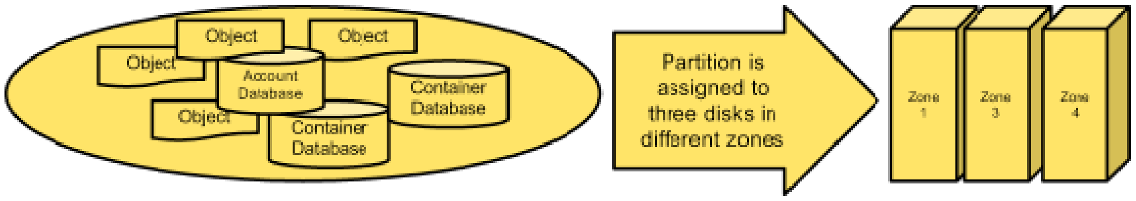
A partition is a collection of stored data, including account databases, container databases, and objects. Partitions are core to the replication system.

Think of a partition as a bin moving throughout a fulfillment center warehouse. Individual orders get thrown into the bin. The system treats that bin as a cohesive entity as it moves throughout the system. A bin is easier to deal with than many little things. It makes for fewer moving parts throughout the system.

System replicators and object uploads/downloads operate on partitions. As the system scales up, its behavior continues to be predictable because the number of partitions is a fixed number.

Implementing a partition is conceptually simple, a partition is just a directory sitting on a disk with a corresponding hash table of what it contains.

Figure 5.6. Partitions



Replicators

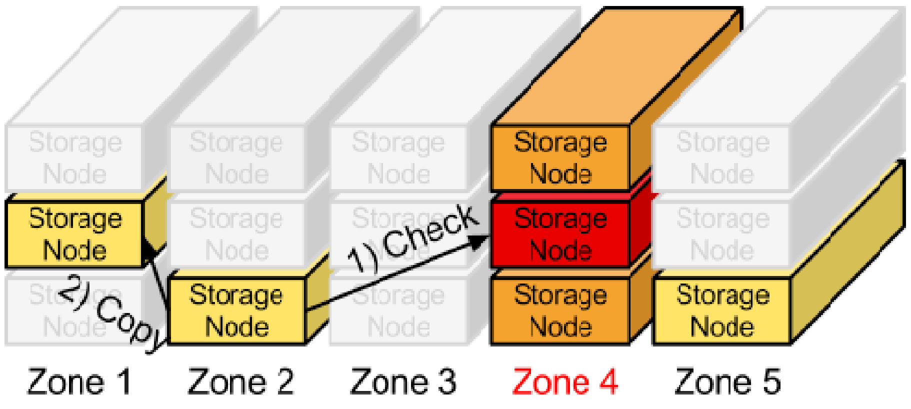
In order to ensure that there are three copies of the data everywhere, replicators continuously examine each partition. For each local partition, the replicator compares it against the replicated copies in the other zones to see if there are any differences.

The replicator knows if replication needs to take place by examining hashes. A hash file is created for each partition, which contains hashes of each directory in the partition. Each of the three hash files is compared. For a given partition, the hash files for each of the partition's copies are compared. If the hashes are different, then it is time to replicate, and the directory that needs to be replicated is copied over.

This is where partitions come in handy. With fewer things in the system, larger chunks of data are transferred around (rather than lots of little TCP connections, which is inefficient) and there is a consistent number of hashes to compare.

The cluster eventually has a consistent behavior where the newest data has a priority.

Figure 5.7. Replication



If a zone goes down, one of the nodes containing a replica notices and proactively copies data to a handoff location.

Use cases

The following sections show use cases for object uploads and downloads and introduce the components.

Upload

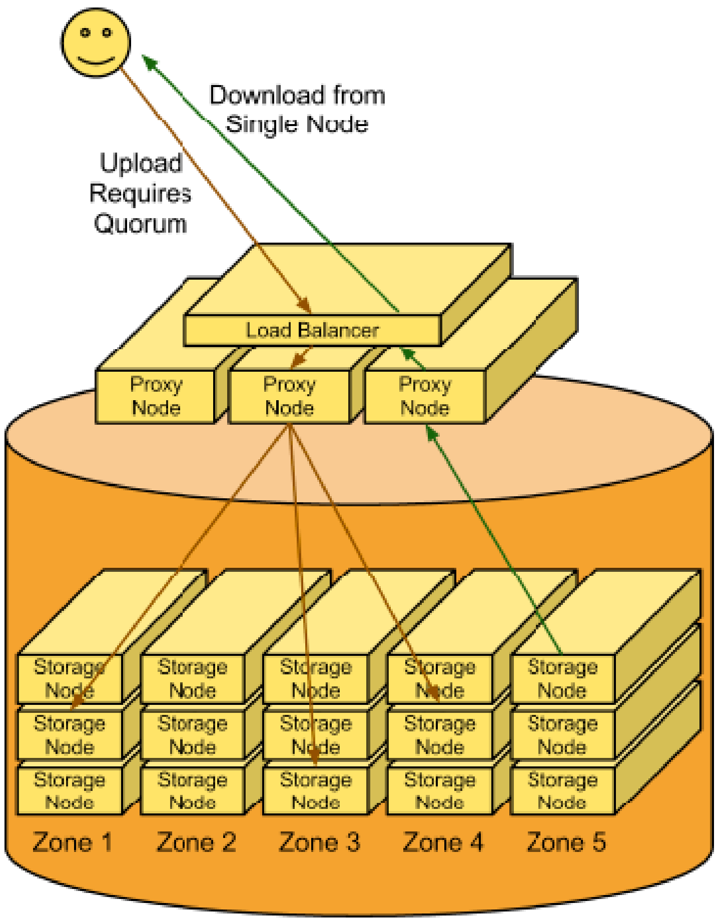
A client uses the REST API to make a HTTP request to PUT an object into an existing container. The cluster receives the request. First, the system must figure out where the data is going to go. To do this, the account name, container name, and object name are all used to determine the partition where this object should live.

Then a lookup in the ring figures out which storage nodes contain the partitions in question.

The data is then sent to each storage node where it is placed in the appropriate partition. At least two of the three writes must be successful before the client is notified that the upload was successful.

Next, the container database is updated asynchronously to reflect that there is a new object in it.

Figure 5.8. Object Storage in use



Download

A request comes in for an account/container/object. Using the same consistent hashing, the partition name is generated. A lookup in the ring reveals which storage nodes contain that partition. A request is made to one of the storage nodes to fetch the object and, if that fails, requests are made to the other nodes.

Ring-builder

Use the swift-ring-builder utility to build and manage rings. This utility assigns partitions to devices and writes an optimized Python structure to a gzipped, serialized file on disk for transmission to the servers. The server processes occasionally check the modification time of the file and reload in-memory copies of the ring structure as needed. If you use a slightly older version of the ring, one of the three replicas for a partition subset will be incorrect because of the way the ring-builder manages changes to the ring. You can work around this issue.

The ring-builder also keeps its own builder file with the ring information and additional data required to build future rings. It is very important to keep multiple backup copies of these builder files. One option is to copy the builder files out to every server while copying the ring files themselves. Another is to upload the builder files into the cluster itself. If you lose the builder file, you have to create a new ring from scratch. Nearly all partitions would be assigned to different devices and, therefore, nearly all of the stored data would have to be replicated to new locations. So, recovery from a builder file loss is possible, but data would be unreachable for an extended time.

Ring data structure

The ring data structure consists of three top level fields: a list of devices in the cluster, a list of lists of device ids indicating partition to device assignments, and an integer indicating the number of bits to shift an MD5 hash to calculate the partition for the hash.

Partition assignment list

This is a list of `array('H')` of devices ids. The outermost list contains an `array('H')` for each replica. Each `array('H')` has a length equal to the partition count for the ring. Each integer in the `array('H')` is an index into the above list of devices. The partition list is known internally to the Ring class as `_replica2part2dev_id`.

So, to create a list of device dictionaries assigned to a partition, the Python code would look like:

```
devices = [self.devs[part2dev_id[partition]] for
part2dev_id in self._replica2part2dev_id]
```

That code is a little simplistic because it does not account for the removal of duplicate devices. If a ring has more replicas than devices, a partition will have more than one replica on a device.

`array('H')` is used for memory conservation as there may be millions of partitions.

Replica counts

To support the gradual change in replica counts, a ring can have a real number of replicas and is not restricted to an integer number of replicas.

A fractional replica count is for the whole ring and not for individual partitions. It indicates the average number of replicas for each partition. For example, a replica count of 3.2 means that 20 percent of partitions have four replicas and 80 percent have three replicas.

The replica count is adjustable.

Example:

```
$ swift-ring-builder account.builder set_replicas 4
$ swift-ring-builder account.builder rebalance
```

You must rebalance the replica ring in globally distributed clusters. Operators of these clusters generally want an equal number of replicas and regions. Therefore, when an operator adds or removes a region, the operator adds or removes a replica. Removing unneeded replicas saves on the cost of disks.

You can gradually increase the replica count at a rate that does not adversely affect cluster performance.

For example:

```
$ swift-ring-builder object.builder set_replicas 3.01
$ swift-ring-builder object.builder rebalance
<distribute rings and wait>...

$ swift-ring-builder object.builder set_replicas 3.02
$ swift-ring-builder object.builder rebalance
<creatdistribute rings and wait>...
```

Changes take effect after the ring is rebalanced. Therefore, if you intend to change from 3 replicas to 3.01 but you accidentally type `2.01`, no data is lost.

Additionally, **swift-ring-builder *`X.builder`* create** can now take a decimal argument for the number of replicas.

Partition shift value

The partition shift value is known internally to the Ring class as `_part_shift`. This value is used to shift an MD5 hash to calculate the partition where the data for that hash should reside. Only the top four bytes of the hash is used in this process. For example, to compute the partition for the `/account/container/object` path, the Python code might look like the following code:

```
partition = unpack_from('>I',
md5('/account/container/object').digest())[0] >>
self._part_shift
```

For a ring generated with part_power P, the partition shift value is `32 - P`.

Build the ring

The ring builder process includes these high-level steps:

1. The utility calculates the number of partitions to assign to each device based on the weight of the device. For example, for a partition at the power of 20, the ring has 1,048,576 partitions. One thousand devices of equal weight will each want 1,048.576 partitions. The devices are sorted by the number of partitions they desire and kept in order throughout the initialization process.

   Note

   Each device is also assigned a random tiebreaker value that is used when two devices desire the same number of partitions. This tiebreaker is not stored on disk anywhere, and so two different rings created with the same parameters will have different partition assignments. For repeatable partition assignments, `RingBuilder.rebalance()` takes an optional seed value that seeds the Python pseudo-random number generator.

2. The ring builder assigns each partition replica to the device that requires most partitions at that point while keeping it as far away as possible from other replicas. The ring

builder prefers to assign a replica to a device in a region that does not already have a replica. If no such region is available, the ring builder searches for a device in a different zone, or on a different server. If it does not find one, it looks for a device with no replicas. Finally, if all options are exhausted, the ring builder assigns the replica to the device that has the fewest replicas already assigned.

Note

The ring builder assigns multiple replicas to one device only if the ring has fewer devices than it has replicas.

3. When building a new ring from an old ring, the ring builder recalculates the desired number of partitions that each device wants.

4. The ring builder unassigns partitions and gathers these partitions for reassignment, as follows:

   - The ring builder unassigns any assigned partitions from any removed devices and adds these partitions to the gathered list.

   - The ring builder unassigns any partition replicas that can be spread out for better durability and adds these partitions to the gathered list.

   - The ring builder unassigns random partitions from any devices that have more partitions than they need and adds these partitions to the gathered list.

5. The ring builder reassigns the gathered partitions to devices by using a similar method to the one described previously.

6. When the ring builder reassigns a replica to a partition, the ring builder records the time of the reassignment. The ring builder uses this value when it gathers partitions for reassignment so that no partition is moved twice in a configurable amount of time. The RingBuilder class knows this configurable amount of time as `min_part_hours`. The ring builder ignores this restriction for replicas of partitions on removed devices because removal of a device happens on device failure only, and reassignment is the only choice.

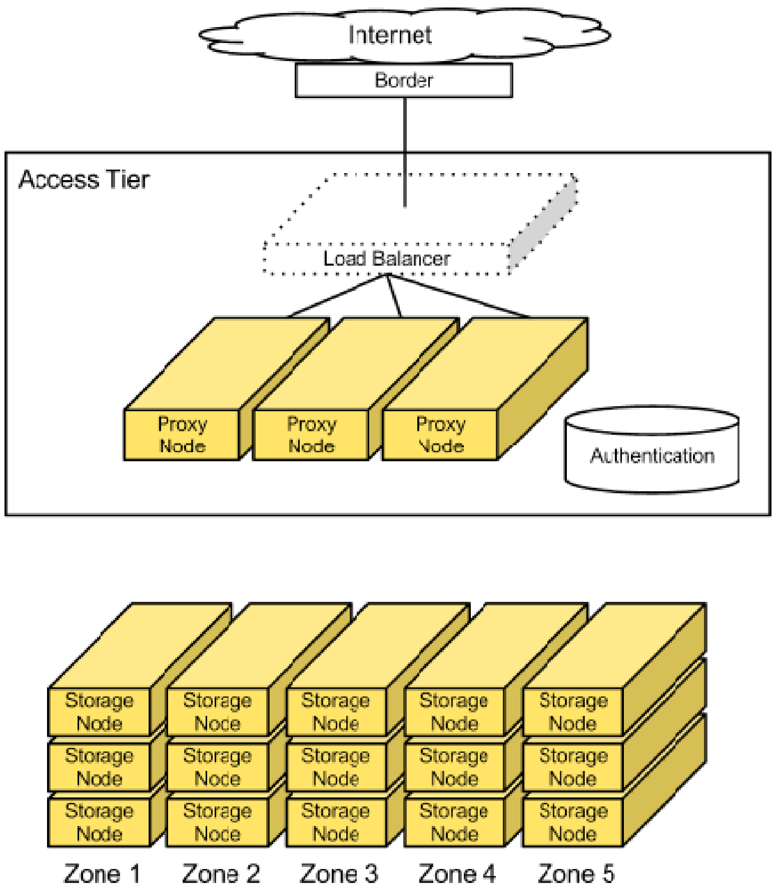
Theses steps do not always perfectly rebalance a ring due to the random nature of gathering partitions for reassignment. To help reach a more balanced ring, the rebalance process is repeated until near perfect (less than 1 percent off) or when the balance does not improve by at least 1 percent (indicating we probably cannot get perfect balance due to wildly imbalanced zones or too many partitions recently moved).

Cluster architecture

Access tier

Large-scale deployments segment off an access tier, which is considered the Object Storage system's central hub. The access tier fields the incoming API requests from clients and moves data in and out of the system. This tier consists of front-end load balancers, ssl-terminators, and authentication services. It runs the (distributed) brain of the Object Storage system: the proxy server processes.

Figure 5.9. Object Storage architecture



Because access servers are collocated in their own tier, you can scale out read/write access regardless of the storage capacity. For example, if a cluster is on the public Internet, requires SSL termination, and has a high demand for data access, you can provision many access servers. However, if the cluster is on a private network and used primarily for archival purposes, you need fewer access servers.

Since this is an HTTP addressable storage service, you may incorporate a load balancer into the access tier.

Typically, the tier consists of a collection of 1U servers. These machines use a moderate amount of RAM and are network I/O intensive. Since these systems field each incoming API request, you should provision them with two high-throughput (10GbE) interfaces - one for the incoming "front-end" requests and the other for the "back-end" access to the object storage nodes to put and fetch data.

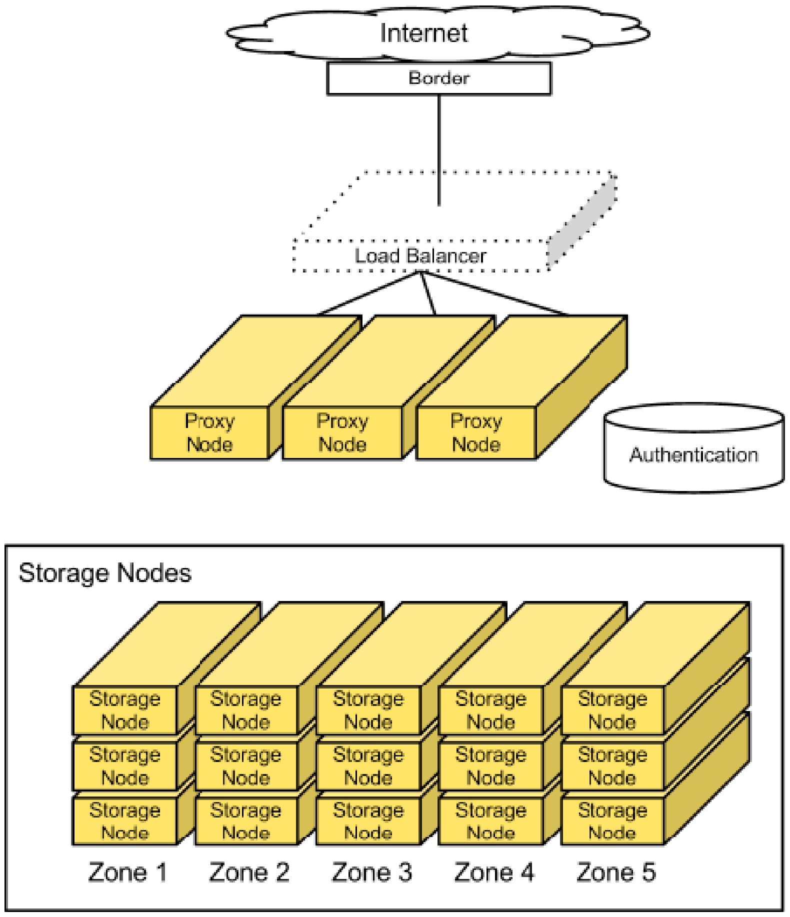
Factors to consider

For most publicly facing deployments as well as private deployments available across a wide-reaching corporate network, you use SSL to encrypt traffic to the client. SSL adds significant processing load to establish sessions between clients, which is why you have to provision more capacity in the access layer. SSL may not be required for private deployments on trusted networks.

Storage nodes

In most configurations, each of the five zones should have an equal amount of storage capacity. Storage nodes use a reasonable amount of memory and CPU. Metadata needs to be readily available to return objects quickly. The object stores run services not only to field incoming requests from the access tier, but to also run replicators, auditors, and reapers. You can provision object stores provisioned with single gigabit or 10 gigabit network interface depending on the expected workload and desired performance.

Figure 5.10. Object Storage (Swift)



Currently, a 2 TB or 3 TB SATA disk delivers good performance for the price. You can use desktop-grade drives if you have responsive remote hands in the datacenter and enterprise-grade drives if you don't.

Factors to consider

You should keep in mind the desired I/O performance for single-threaded requests . This system does not use RAID, so a single disk handles each request for an object. Disk performance impacts single-threaded response rates.

To achieve apparent higher throughput, the object storage system is designed to handle concurrent uploads/downloads. The network I/O capacity (1GbE, bonded 1GbE pair, or 10GbE) should match your desired concurrent throughput needs for reads and writes.

Replication

Because each replica in Object Storage functions independently and clients generally require only a simple majority of nodes to respond to consider an operation successful, transient failures like network partitions can quickly cause replicas to diverge. These differences are eventually reconciled by asynchronous, peer-to-peer replicator processes. The replicator processes traverse their local file systems and concurrently perform operations in a manner that balances load across physical disks.

Replication uses a push model, with records and files generally only being copied from local to remote replicas. This is important because data on the node might not belong there (as in the case of hand offs and ring changes), and a replicator cannot know which data it should pull in from elsewhere in the cluster. Any node that contains data must ensure that data gets to where it belongs. The ring handles replica placement.

To replicate deletions in addition to creations, every deleted record or file in the system is marked by a tombstone. The replication process cleans up tombstones after a time period known as the *consistency window*. This window defines the duration of the replication and how long transient failure can remove a node from the cluster. Tombstone cleanup must be tied to replication to reach replica convergence.

If a replicator detects that a remote drive has failed, the replicator uses the `get_more_nodes` interface for the ring to choose an alternate node with which to synchronize. The replicator can maintain desired levels of replication during disk failures, though some replicas might not be in an immediately usable location.

Note

The replicator does not maintain desired levels of replication when failures such as entire node failures occur; most failures are transient.

The main replication types are:

- **Database replication**. Replicates containers and objects.
- **Object replication**. Replicates object data.

Database replication

Database replication completes a low-cost hash comparison to determine whether two replicas already match. Normally, this check can quickly verify that most databases in the system are already synchronized. If the hashes differ, the replicator synchronizes the databases by sharing records added since the last synchronization point.

This synchronization point is a high water mark that notes the last record at which two databases were known to be synchronized, and is stored in each database as a tuple of the remote database ID and record ID. Database IDs are unique across all replicas of the database, and record IDs are monotonically increasing integers. After all new records are pushed to the remote database, the entire synchronization table of the local database is pushed, so the remote database can guarantee that it is synchronized with everything with which the local database was previously synchronized.

If a replica is missing, the whole local database file is transmitted to the peer by using rsync(1) and is assigned a new unique ID.

In practice, database replication can process hundreds of databases per concurrency setting per second (up to the number of available CPUs or disks) and is bound by the number of database transactions that must be performed.

Object replication

The initial implementation of object replication performed an rsync to push data from a local partition to all remote servers where it was expected to reside. While this worked at small scale, replication times skyrocketed once directory structures could no longer be held in RAM. This scheme was modified to save a hash of the contents for each suffix directory to a per-partition hashes file. The hash for a suffix directory is no longer valid when the contents of that suffix directory is modified.

The object replication process reads in hash files and calculates any invalidated hashes. Then, it transmits the hashes to each remote server that should hold the partition, and only suffix directories with differing hashes on the remote server are rsynced. After pushing files to the remote server, the replication process notifies it to recalculate hashes for the rsynced suffix directories.

The number of uncached directories that object replication must traverse, usually as a result of invalidated suffix directory hashes, impedes performance. To provide acceptable replication speeds, object replication is designed to invalidate around 2 percent of the hash space on a normal node each day.

Object Storage monitoring

Excerpted from a blog post by Darrell Bishop

An OpenStack Object Storage cluster is a collection of many daemons that work together across many nodes. With so many different components, you must be able to tell what is going on inside the cluster. Tracking server-level metrics like CPU utilization, load, memory consumption, disk usage and utilization, and so on is necessary, but not sufficient.

What are different daemons are doing on each server? What is the volume of object replication on node8? How long is it taking? Are there errors? If so, when did they happen?

In such a complex ecosystem, you can use multiple approaches to get the answers to these questions. This section describes several approaches.

Swift Recon

The Swift Recon middleware (see http://swift.openstack.org/admin_guide.html#cluster-telemetry-and-monitoring) provides general machine statistics, such as load average, socket statistics, `/proc/meminfo` contents, and so on, as well as Swift-specific metrics:

- The MD5 sum of each ring file.
- The most recent object replication time.

- Count of each type of quarantined file: Account, container, or object.
- Count of “async_pendings” (deferred container updates) on disk.

Swift Recon is middleware that is installed in the object servers pipeline and takes one required option: A local cache directory. To track `async_pendings`, you must set up an additional cron job for each object server. You access data by either sending HTTP requests directly to the object server or using the **swift-recon** command-line client.

There are some good Object Storage cluster statistics but the general server metrics overlap with existing server monitoring systems. To get the Swift-specific metrics into a monitoring system, they must be polled. Swift Recon essentially acts as a middleware metrics collector. The process that feeds metrics to your statistics system, such as `collectd` and `gmond`, probably already runs on the storage node. So, you can choose to either talk to Swift Recon or collect the metrics directly.

Swift-Informant

Florian Hines developed the Swift-Informant middleware (see http://pandemicsyn.posterous.com/swift-informant-statsd-getting-realtime-telem) to get real-time visibility into Object Storage client requests. It sits in the pipeline for the proxy server, and after each request to the proxy server, sends three metrics to a StatsD server (see http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/):

- A counter increment for a metric like `obj.GET.200` or `cont.PUT.404`.
- Timing data for a metric like `acct.GET.200` or `obj.GET.200`. [The README says the metrics look like `duration.acct.GET.200`, but I do not see the `duration` in the code. I am not sure what the Etsy server does but our StatsD server turns timing metrics into five derivative metrics with new segments appended, so it probably works as coded. The first metric turns into `acct.GET.200.lower`, `acct.GET.200.upper`, `acct.GET.200.mean`, `acct.GET.200.upper_90`, and `acct.GET.200.count`].
- A counter increase by the bytes transferred for a metric like `tfer.obj.PUT.201`.

This is good for getting a feel for the quality of service clients are experiencing with the timing metrics, as well as getting a feel for the volume of the various permutations of request server type, command, and response code. Swift-Informant also requires no change to core Object Storage code because it is implemented as middleware. However, it gives you no insight into the workings of the cluster past the proxy server. If the responsiveness of one storage node degrades, you can only see that some of your requests are bad, either as high latency or error status codes. You do not know exactly why or where that request tried to go. Maybe the container server in question was on a good node but the object server was on a different, poorly-performing node.

Statsdlog

Florian’s Statsdlog project increments StatsD counters based on logged events. Like Swift-Informant, it is also non-intrusive, but statsdlog can track events from all Object Storage daemons, not just proxy-server. The daemon listens to a UDP stream of syslog messages and StatsD counters are incremented when a log line matches a regular expression. Metric names are mapped to regex match patterns in a JSON file, allowing flexible configuration of what metrics are extracted from the log stream.

Currently, only the first matching regex triggers a StatsD counter increment, and the counter is always incremented by one. There is no way to increment a counter by more than one or send timing data to StatsD based on the log line content. The tool could be extended to handle more metrics for each line and data extraction, including timing data. But a coupling would still exist between the log textual format and the log parsing regexes, which would themselves be more complex to support multiple matches for each line and data extraction. Also, log processing introduces a delay between the triggering event and sending the data to StatsD. It would be preferable to increment error counters where they occur and send timing data as soon as it is known to avoid coupling between a log string and a parsing regex and prevent a time delay between events and sending data to StatsD.

The next section describes another method for gathering Object Storage operational metrics.

Swift StatsD logging

StatsD (see http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/) was designed for application code to be deeply instrumented; metrics are sent in real-time by the code that just noticed or did something. The overhead of sending a metric is extremely low: a `sendto` of one UDP packet. If that overhead is still too high, the StatsD client library can send only a random portion of samples and StatsD approximates the actual number when flushing metrics upstream.

To avoid the problems inherent with middleware-based monitoring and after-the-fact log processing, the sending of StatsD metrics is integrated into Object Storage itself. The submitted change set (see https://review.openstack.org/#change,6058) currently reports 124 metrics across 15 Object Storage daemons and the tempauth middleware. Details of the metrics tracked are in the Administrator's Guide.

The sending of metrics is integrated with the logging framework. To enable, configure `log_statsd_host` in the relevant config file. You can also specify the port and a default sample rate. The specified default sample rate is used unless a specific call to a statsd logging method (see the list below) overrides it. Currently, no logging calls override the sample rate, but it is conceivable that some metrics may require accuracy (sample_rate == 1) while others may not.

```
[DEFAULT]
    ...
log_statsd_host = 127.0.0.1
log_statsd_port = 8125
log_statsd_default_sample_rate = 1
```

Then the LogAdapter object returned by `get_logger()`, usually stored in `self.logger`, has these new methods:

- `set_statsd_prefix(self, prefix)` Sets the client library stat prefix value which gets prefixed to every metric. The default prefix is the “name” of the logger (such as, . “object-server”, “container-auditor”, etc.). This is currently used to turn “proxy-server” into one of “proxy-server.Account”, “proxy-server.Container”, or “proxy-server.Object” as soon as the Controller object is determined and instantiated for the request.

- `update_stats(self, metric, amount, sample_rate=1)` Increments the supplied metric by the given amount. This is used when you need to add or subtract more that one from a counter, like incrementing "suffix.hashes" by the number of computed hashes in the object replicator.

- `increment(self, metric, sample_rate=1)` Increments the given counter metric by one.
- `decrement(self, metric, sample_rate=1)` Lowers the given counter metric by one.
- `timing(self, metric, timing_ms, sample_rate=1)` Record that the given metric took the supplied number of milliseconds.
- `timing_since(self, metric, orig_time, sample_rate=1)` Convenience method to record a timing metric whose value is "now" minus an existing timestamp.

Note that these logging methods may safely be called anywhere you have a logger object. If StatsD logging has not been configured, the methods are no-ops. This avoids messy conditional logic each place a metric is recorded. These example usages show the new logging methods:

```
# swift/obj/replicator.py
def update(self, job):
    # ...
    begin = time.time()
    try:
        hashed, local_hash = tpool.execute(tpooled_get_hashes, job['path'],
                do_listdir=(self.replication_count % 10) == 0,
                reclaim_age=self.reclaim_age)
        # See tpooled_get_hashes "Hack".
        if isinstance(hashed, BaseException):
            raise hashed
        self.suffix_hash += hashed
        self.logger.update_stats('suffix.hashes', hashed)
        # ...
    finally:
        self.partition_times.append(time.time() - begin)
        self.logger.timing_since('partition.update.timing', begin)
```

```
# swift/container/updater.py
def process_container(self, dbfile):
    # ...
    start_time = time.time()
    # ...
        for event in events:
            if 200 <= event.wait() < 300:
                successes += 1
            else:
                failures += 1
        if successes > failures:
            self.logger.increment('successes')
            # ...
        else:
            self.logger.increment('failures')
            # ...
        # Only track timing data for attempted updates:
        self.logger.timing_since('timing', start_time)
    else:
        self.logger.increment('no_changes')
        self.no_changes += 1
```

The development team of StatsD wanted to use the pystatsd client library (not to be confused with a similar-looking project also hosted on GitHub), but the released version on PyPi was missing two desired features the latest version in GitHub had: the ability to configure a metrics prefix in the client object and a convenience method for sending timing data between "now" and a "start" timestamp you already have. So they just implemented a simple StatsD client library from scratch with the same interface. This has the nice fringe benefit of not introducing another external library dependency into Object Storage.

Troubleshoot Object Storage

For Object Storage, everything is logged in `/var/log/syslog` (or messages on some distros). Several settings enable further customization of logging, such as `log_name`, `log_facility`, and `log_level`, within the object server configuration files.

Drive failure

In the event that a drive has failed, the first step is to make sure the drive is unmounted. This will make it easier for Object Storage to work around the failure until it has been resolved. If the drive is going to be replaced immediately, then it is just best to replace the drive, format it, remount it, and let replication fill it up.

If the drive can't be replaced immediately, then it is best to leave it unmounted, and remove the drive from the ring. This will allow all the replicas that were on that drive to be replicated elsewhere until the drive is replaced. Once the drive is replaced, it can be re-added to the ring.

You can look at error messages in `/var/log/kern.log` for hints of drive failure.

Server failure

If a server is having hardware issues, it is a good idea to make sure the Object Storage services are not running. This will allow Object Storage to work around the failure while you troubleshoot.

If the server just needs a reboot, or a small amount of work that should only last a couple of hours, then it is probably best to let Object Storage work around the failure and get the machine fixed and back online. When the machine comes back online, replication will make sure that anything that is missing during the downtime will get updated.

If the server has more serious issues, then it is probably best to remove all of the server's devices from the ring. Once the server has been repaired and is back online, the server's devices can be added back into the ring. It is important that the devices are reformatted before putting them back into the ring as it is likely to be responsible for a different set of partitions than before.

Detect failed drives

It has been our experience that when a drive is about to fail, error messages will spew into /var/log/kern.log. There is a script called swift-drive-audit that can be run via cron to watch for bad drives. If errors are detected, it will unmount the bad drive, so that Object Storage can work around it. The script takes a configuration file with the following settings:

Table 5.1. Description of configuration options for `[drive-audit]` in `drive-audit.conf-sample`

Configuration option = Default value	Description
device_dir = /srv/node	Directory devices are mounted under
log_facility = LOG_LOCAL0	Syslog log facility
log_level = INFO	Logging level
log_address = /dev/log	Location where syslog sends the logs to
minutes = 60	Number of minutes to look back in `/var/log/kern.log`
error_limit = 1	Number of errors to find before a device is unmounted
log_file_pattern = /var/log/kern*	Location of the log file with globbing pattern to check against device errors locate device blocks with errors in the log file
regex_pattern_1 = \berror\b.*\b(dm-[0-9]{1,2}\d?)\b	No help text available for this option.

This script has only been tested on Ubuntu 10.04, so if you are using a different distro or OS, some care should be taken before using in production.

Emergency recovery of ring builder files

You should always keep a backup of Swift ring builder files. However, if an emergency occurs, this procedure may assist in returning your cluster to an operational state.

Using existing Swift tools, there is no way to recover a builder file from a `ring.gz` file. However, if you have a knowledge of Python, it is possible to construct a builder file that is pretty close to the one you have lost. The following is what you will need to do.

Warning

This procedure is a last-resort for emergency circumstances. It requires knowledge of the swift python code and may not succeed.

First, load the ring and a new ringbuilder object in a Python REPL:

```
>>> from swift.common.ring import RingData, RingBuilder
>>> ring = RingData.load('/path/to/account.ring.gz')
```

Now, start copying the data we have in the ring into the builder.

```

>>> import math
>>> partitions = len(ring._replica2part2dev_id[0])
>>> replicas = len(ring._replica2part2dev_id)

>>> builder = RingBuilder(int(Math.log(partitions, 2)), replicas, 1)
>>> builder.devs = ring.devs
>>> builder._replica2part2dev = ring.replica2part2dev_id
>>> builder._last_part_moves_epoch = 0
>>> builder._last_part_moves = array('B', (0 for _ in xrange(self.parts)))
>>> builder._set_parts_wanted()
>>> for d in builder._iter_devs():
```

```
            d['parts'] = 0
>>> for p2d in builder._replica2part2dev:
            for dev_id in p2d:
                builder.devs[dev_id]['parts'] += 1
```

This is the extent of the recoverable fields. For `min_part_hours` you'll either have to remember what the value you used was, or just make up a new one.

```
>>> builder.change_min_part_hours(24) # or whatever you want it to be
```

Try some validation: if this doesn't raise an exception, you may feel some hope. Not too much, though.

```
>>> builder.validate()
```

Save the builder.

```
>>> import pickle
>>> pickle.dump(builder.to_dict(), open('account.builder', 'wb'), protocol=2)
```

You should now have a file called 'account.builder' in the current working directory. Next, run `swift-ring-builder account.builder write_ring` and compare the new account.ring.gz to the account.ring.gz that you started from. They probably won't be byte-for-byte identical, but if you load them up in a REPL and their `_replica2part2dev_id` and `devs` attributes are the same (or nearly so), then you're in good shape.

Next, repeat the procedure for `container.ring.gz` and `object.ring.gz`, and you might get usable builder files.

6. Block Storage

Table of Contents

The OpenStack Block Storage service works through the interaction of a series of daemon processes named `cinder-*` that reside persistently on the host machine or machines. The binaries can all be run from a single node, or spread across multiple nodes. They can also be run on the same node as other OpenStack services.

Introduction to Block Storage

To administer the OpenStack Block Storage service, it is helpful to understand a number of concepts. You must make certain choices when you configure the Block Storage service in OpenStack. The bulk of the options come down to two choices, single node or multi-node install. You can read a longer discussion about storage decisions in Storage Decisions in the *OpenStack Operations Guide*.

OpenStack Block Storage enables you to add extra block-level storage to your OpenStack Compute instances. This service is similar to the Amazon EC2 Elastic Block Storage (EBS) offering.

Manage volumes

The default OpenStack Block Storage service implementation is an iSCSI solution that uses Logical Volume Manager (LVM) for Linux.

Note

The OpenStack Block Storage service is not a shared storage solution like a Storage Area Network (SAN) of NFS volumes, where you can attach a volume to multiple servers. With the OpenStack Block Storage service, you can attach a volume to only one instance at a time.

The OpenStack Block Storage service also provides drivers that enable you to use several vendors' back-end storage devices, in addition to or instead of the base LVM implementation.

This high-level procedure shows you how to create and attach a volume to a server instance.

1. You must configure both OpenStack Compute and the OpenStack Block Storage service through the `cinder.conf` file.
2. Create a volume through the **cinder create** command. This command creates an LV into the volume group (VG) `cinder-volumes`.
3. Attach the volume to an instance through the **nova volume-attach** command. This command creates a unique iSCSI IQN that is exposed to the compute node.
    a. The compute node, which runs the instance, now has an active ISCSI session and new local storage (usually a `/dev/sdX` disk).
    b. libvirt uses that local storage as storage for the instance. The instance gets a new disk (usually a `/dev/vdX` disk).

For this particular walk through, one cloud controller runs `nova-api`, `nova-scheduler`, `nova-objectstore`, `nova-network` and `cinder-*` services. Two additional compute nodes run `nova-compute`. The walk through uses a custom partitioning scheme that carves out 60 GB of space and labels it as LVM. The network uses the `FlatManager` and `NetworkManager` settings for OpenStack Compute (Nova).

The network mode does not interfere with the way cinder works, but you must set up networking for cinder to work. For details, see Chapter 7, “Networking” [167].

To set up Compute to use volumes, ensure that Block Storage is installed along with lvm2. This guide describes how to troubleshoot your installation and back up your Compute volumes.

Boot from volume

In some cases, instances can be stored and run from inside volumes. For information, see the Launch an instance from a volume section in the *OpenStack End User Guide*.

Configure an NFS storage back end

This section explains how to configure OpenStack Block Storage to use NFS storage. You must be able to access the NFS shares from the server that hosts the `cinder` volume service.

Note

The `cinder` volume service is named `openstack-cinder-volume` on the following distributions:

- CentOS
- Fedora
- openSUSE
- Red Hat Enterprise Linux
- SUSE Linux Enterprise

In Ubuntu and Debian distributions, the `cinder` volume service is named `cinder-volume`.

Procedure 6.1. Configure Block Storage to use an NFS storage back end

1. Log in as `root` to the system hosting the `cinder` volume service.

2. Create a text file named `nfsshares` in `/etc/cinder/`.

3. Add an entry to `/etc/cinder/nfsshares` for each NFS share that the `cinder` volume service should use for back end storage. Each entry should be a separate line, and should use the following format:

   ```
   HOST:SHARE
   ```

   Where:

   - *HOST* is the IP address or host name of the NFS server.
   - *SHARE* is the absolute path to an existing and accessible NFS share.

4. Set `/etc/cinder/nfsshares` to be owned by the `root` user and the `cinder` group:

   ```
   # chown root:cinder /etc/cinder/nfsshares
   ```

5. Set `/etc/cinder/nfsshares` to be readable by members of the `cinder` group:

   ```
   # chmod 0640 /etc/cinder/nfsshares
   ```

6. Configure the `cinder` volume service to use the `/etc/cinder/nfsshares` file created earlier. To do so, open the `/etc/cinder/cinder.conf` configuration file and set the `nfs_shares_config` configuration key to `/etc/cinder/nfsshares`.

On distributions that include openstack-config, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
DEFAULT nfs_shares_config /etc/cinder/nfsshares
```

The following distributions include openstack-config:

- CentOS
- Fedora
- openSUSE
- Red Hat Enterprise Linux
- SUSE Linux Enterprise

7. Optionally, provide any additional NFS mount options required in your environment in the `nfs_mount_options` configuration key of `/etc/cinder/cinder.conf`. If your NFS shares do not require any additional mount options (or if you are unsure), skip this step.

   On distributions that include openstack-config, you can configure this by running the following command instead:

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
   DEFAULT nfs_mount_options OPTIONS
   ```

   Replace *`OPTIONS`* with the mount options to be used when accessing NFS shares. See the manual page for NFS for more information on available mount options (**man nfs**).

8. Configure the `cinder` volume service to use the correct volume driver, namely `cinder.volume.drivers.nfs.NfsDriver`. To do so, open the `/etc/cinder/cinder.conf` configuration file and set the `volume_driver` configuration key to `cinder.volume.drivers.nfs.NfsDriver`.

   On distributions that include openstack-config, you can configure this by running the following command instead:

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
   DEFAULT volume_driver cinder.volume.drivers.nfs.NfsDriver
   ```

9. You can now restart the service to apply the configuration.

   To restart the `cinder` volume service on CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, or SUSE Linux Enterprise, run:

   ```
   # service openstack-cinder-volume restart
   ```

   To restart the `cinder` volume service on Ubuntu or Debian, run:

   ```
   # service cinder-volume restart
   ```

Note

The `nfs_sparsed_volumes` configuration key determines whether volumes are created as sparse files and grown as needed or fully allocated up front. The default and recommended value is `true`, which ensures volumes are initially created as sparse files.

Setting `nfs_sparsed_volumes` to `false` will result in volumes being fully allocated at the time of creation. This leads to increased delays in volume creation.

However, should you choose to set `nfs_sparsed_volumes` to `false`, you can do so directly in `/etc/cinder/cinder.conf`.

On distributions that include openstack-config, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
DEFAULT nfs_sparsed_volumes false
```

Important

If a client host has SELinux enabled, the `virt_use_nfs` Boolean should also be enabled if the host requires access to NFS volumes on an instance. To enable this Boolean, run the following command as the `root` user:

```
# setsebool -P virt_use_nfs on
```

This command also makes the Boolean persistent across reboots. Run this command on all client hosts that require access to NFS volumes on an instance. This includes all Compute nodes.

Configure a GlusterFS back end

This section explains how to configure OpenStack Block Storage to use GlusterFS as a back end. You must be able to access the GlusterFS shares from the server that hosts the `cinder` volume service.

Note

The `cinder` volume service is named `openstack-cinder-volume` on the following distributions:

- CentOS
- Fedora
- openSUSE
- Red Hat Enterprise Linux
- SUSE Linux Enterprise

In Ubuntu and Debian distributions, the `cinder` volume service is named `cinder-volume`.

Mounting GlusterFS volumes requires utilities and libraries from the glusterfs-fuse package. This package must be installed on all systems that will access volumes backed by GlusterFS.

Note

The utilities and libraries required for mounting GlusterFS volumes on Ubuntu and Debian distributions are available from the glusterfs-client package instead.

For information on how to install and configure GlusterFS, refer to the GlusterDocumentation page.

Procedure 6.2. Configure GlusterFS for OpenStack Block Storage

The GlusterFS server must also be configured accordingly in order to allow OpenStack Block Storage to use GlusterFS shares:

1. Log in as `root` to the GlusterFS server.

2. Set each Gluster volume to use the same UID and GID as the `cinder` user:

   ```
   # gluster volume set VOL_NAME storage.owner-uid cinder-uid
   # gluster volume set VOL_NAME storage.owner-gid cinder-gid
   ```

   Where:

   - *VOL_NAME* is the Gluster volume name.
   - *cinder-uid* is the UID of the `cinder` user.
   - *cinder-gid* is the GID of the `cinder` user.

   Note

   The default UID and GID of the `cinder` user is `165` on most distributions.

3. Configure each Gluster volume to accept `libgfapi` connections. To do this, set each Gluster volume to allow insecure ports:

   ```
   # gluster volume set VOL_NAME server.allow-insecure on
   ```

4. Enable client connections from unprivileged ports. To do this, add the following line to `/etc/glusterfs/glusterd.vol`:

   ```
   option rpc-auth-allow-insecure on
   ```

5. Restart the `glusterd` service:

   ```
   # service glusterd restart
   ```

Procedure 6.3. Configure Block Storage to use a GlusterFS back end

After you configure the GlusterFS service, complete these steps:

1. Log in as `root` to the system hosting the cinder volume service.

2. Create a text file named `glusterfs` in `/etc/cinder/`.

3. Add an entry to `/etc/cinder/glusterfs` for each GlusterFS share that OpenStack Block Storage should use for back end storage. Each entry should be a separate line, and should use the following format:

   ```
   HOST:/VOL_NAME
   ```

   Where:

   - *`HOST`* is the IP address or host name of the Red Hat Storage server.
   - *`VOL_NAME`* is the name an existing and accessible volume on the GlusterFS server.

   Optionally, if your environment requires additional mount options for a share, you can add them to the share's entry:

   ```
   HOST:/VOL_NAME -o OPTIONS
   ```

   Replace *`OPTIONS`* with a comma-separated list of mount options.

4. Set `/etc/cinder/glusterfs` to be owned by the `root` user and the `cinder` group.

   ```
   # chown root:cinder /etc/cinder/glusterfs
   ```

5. Set `/etc/cinder/glusterfs` to be readable by members of the `cinder` group:

   ```
   # chmod 0640 FILE
   ```

6. Configure OpenStack Block Storage to use the `/etc/cinder/glusterfs` file created earlier. To do so, open the `/etc/cinder/cinder.conf` configuration file and set the `glusterfs_shares_config` configuration key to `/etc/cinder/glusterfs`.

   On distributions that include openstack-config, you can configure this by running the following command instead:

   ```
   # openstack-config --set /etc/cinder/cinder.conf \
   DEFAULT glusterfs_shares_config /etc/cinder/glusterfs
   ```

   The following distributions include openstack-config:

   - CentOS
   - Fedora
   - openSUSE
   - Red Hat Enterprise Linux
   - SUSE Linux Enterprise

7. Configure OpenStack Block Storage to use the correct volume driver, namely `cinder.volume.drivers.glusterfs`. To do so, open the `/etc/cinder/`

`cinder.conf` configuration file and set the `volume_driver` configuration key to `cinder.volume.drivers.glusterfs`.

On distributions that include openstack-config, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
DEFAULT volume_driver cinder.volume.drivers.glusterfs.GlusterfsDriver
```

8. You can now restart the service to apply the configuration.

   To restart the `cinder` volume service on CentOS, Fedora, openSUSE, RedHat Enterprise Linux, or SUSE Linux Enterprise, run:

   ```
   # service openstack-cinder-volume restart
   ```

   To restart the `cinder` volume service on Ubuntu or Debian, run:

   ```
   # service cinder-volume restart
   ```

OpenStack Block Storage is now configured to use a GlusterFS back end.

Note

In `/etc/cinder/cinder.conf`, the `glusterfs_sparsed_volumes` configuration key determines whether volumes are created as sparse files and grown as needed or fully allocated up front. The default and recommended value of this key is `true`, which ensures volumes are initially created as sparse files.

Setting `glusterfs_sparsed_volumes` to `false` will result in volumes being fully allocated at the time of creation. This leads to increased delays in volume creation.

However, should you choose to set `glusterfs_sparsed_volumes` to `false`, you can do so directly in `/etc/cinder/cinder.conf`.

On distributions that include openstack-config, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
DEFAULT glusterfs_sparsed_volumes false
```

Important

If a client host has SELinux enabled, the `virt_use_fusefs` Boolean should also be enabled if the host requires access to GlusterFS volumes on an instance. To enable this Boolean, run the following command as the `root` user:

```
# setsebool -P virt_use_fusefs on
```

This command also makes the Boolean persistent across reboots. Run this command on all client hosts that require access to GlusterFS volumes on an instance. This includes all compute nodes.

Configure a multiple-storage back-end

With multiple storage back-ends configured, you can create several back-end storage solutions serving the same OpenStack Compute configuration. Basically, multi back-end launches one `cinder-volume` for each back-end.

In a multi back-end configuration, each back-end has a name (`volume_backend_name`). Several back-ends can have the same name. In that case, the scheduler properly decides which back-end the volume has to be created in.

The name of the back-end is declared as an extra-specification of a volume type (such as, `volume_backend_name=LVM_iSCSI`). When a volume is created, the scheduler chooses an appropriate back-end to handle the request, according to the volume type specified by the user.

Enable multi back-end

To enable a multi back-end configuration, you must set the `enabled_backends` flag in the `cinder.conf` file. This flag defines the names (separated by a comma) of the configuration groups for the different back-ends: one name is associated to one configuration group for a back-end (such as, `[lvmdriver-1]`).

Note

The configuration group name is not related to the `volume_backend_name`.

The options for a configuration group must be defined in the group (or default options are used). All the standard Block Storage configuration options (`volume_group`, `volume_driver`, and so on) might be used in a configuration group. Configuration values in the `[DEFAULT]` configuration group are not used.

These examples show three back-ends:

```
# a list of back-ends that are served by this compute node
enabled_backends=lvmdriver-1,lvmdriver-2,lvmdriver-3
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
[lvmdriver-2]
volume_group=cinder-volumes-2
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
[lvmdriver-3]
volume_group=cinder-volumes-3
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI_b
```

In this configuration, `lvmdriver-1` and `lvmdriver-2` have the same `volume_backend_name`. If a volume creation requests the `LVM_iSCSI` back-end name, the scheduler uses the capacity filter scheduler to choose the most suitable driver, which is either `lvmdriver-1` or `lvmdriver-2`. The capacity filter scheduler is enabled by default. The next section provides more information. In addition, this example presents a `lvmdriver-3` back-end.

Configure Block Storage scheduler multi back-end

You must enable the `filter_scheduler` option to use multi back-end. Filter scheduler acts in two steps:

1. The filter scheduler filters the available back-ends. By default, `AvailabilityZoneFilter`, `CapacityFilter` and `CapabilitiesFilter` are enabled.

2. The filter scheduler weighs the previously filtered back-ends. By default, `CapacityWeigher` is enabled. The `CapacityWeigher` attributes higher scores to back-ends with the most available.

The scheduler uses the filtering and weighing process to pick the best back-end to handle the request, and explicitly creates volumes on specific back-ends through the use of volume types.

Volume type

Before using it, a volume type has to be declared to Block Storage. This can be done by the following command:

```
$ cinder --os-username admin --os-tenant-name admin type-create lvm
```

Then, an extra-specification has to be created to link the volume type to a back-end name. Run this command:

```
$ cinder --os-username admin --os-tenant-name admin type-key lvm set
 volume_backend_name=LVM_iSCSI
```

This example creates a `lvm` volume type with `volume_backend_name=LVM_iSCSI` as extra-specifications.

Create another volume type:

```
$ cinder --os-username admin --os-tenant-name admin type-create lvm_gold
```

```
$ cinder --os-username admin --os-tenant-name admin type-key lvm_gold set
 volume_backend_name=LVM_iSCSI_b
```

This second volume type is named `lvm_gold` and has `LVM_iSCSI_b` as back-end name.

Note

To list the extra-specifications, use this command:

```
$ cinder --os-username admin --os-tenant-name admin extra-specs-list
```

Note

If a volume type points to a `volume_backend_name` that does not exist in the Block Storage configuration, the `filter_scheduler` returns an error that it cannot find a valid host with the suitable back-end.

Usage

When you create a volume, you must specify the volume type. The extra-specifications of the volume type are used to determine which back-end has to be used.

```
$ cinder create --volume_type lvm --display_name test_multi_backend 1
```

Considering the `cinder.conf` described previously, the scheduler creates this volume on `lvmdriver-1` or `lvmdriver-2`.

```
$ cinder create --volume_type lvm_gold --display_name test_multi_backend 1
```

This second volume is created on `lvmdriver-3`.

Back up Block Storage service disks

While you can use the LVM snapshot to create snapshots, you can also use it to back up your volumes. By using LVM snapshot, you reduce the size of the backup; only existing data is backed up instead of the entire volume.

To back up a volume, you must create a snapshot of it. An LVM snapshot is the exact copy of a logical volume, which contains data in a frozen state. This prevents data corruption, because data cannot be manipulated during the volume creation process. Remember that the volumes created through a **nova volume-create** command exist in an LVM logical volume.

You must also make sure that the operating system is not using the volume, and that all data has been flushed on the guest filesystems. This usually means that those filesystems have to be unmounted during the snapshot creation. They can be mounted again as soon as the logical volume snapshot has been created.

Before you create the snapshot, you must have enough space to save it. As a precaution, you should have at least twice as much space as the potential snapshot size. If insufficient space is available, the snapshot might become corrupted.

For this example, assume that a 100 GB volume named `volume-00000001` was created for an instance while only 4 GB are used. This example uses these commands to back up only those 4 GB:

- **lvm2** command. Directly manipulates the volumes.
- **kpartx** command. Discovers the partition table created inside the instance.
- **tar** command. Creates a minimum-sized backup.
- **sha1sum** command. Calculates the backup checksum to check its consistency.

You can apply this process to volumes of any size.

Procedure 6.4. To back up Block Storage service disks

1. **Create a snapshot of a used volume**

   a. Use this command to list all volumes:

      ```
      # lvdisplay
      ```

b. Create the snapshot; you can do this while the volume is attached to an instance:

```
# lvcreate --size 10G --snapshot --name volume-00000001-snapshot /dev/
cinder-volumes/volume-00000001
```

Use the `--snapshot` configuration option to tell LVM that you want a snapshot of an already existing volume. The command includes the size of the space reserved for the snapshot volume, the name of the snapshot, and the path of an already existing volume. Generally, this path is `/dev/cinder-volumes/$volume_name`.

The size does not have to be the same as the volume of the snapshot. The *`size`* parameter defines the space that LVM reserves for the snapshot volume. As a precaution, the size should be the same as that of the original volume, even if the whole space is not currently used by the snapshot.

c. Run the **lvdisplay** command again to verify the snapshot:

```
--- Logical volume ---
LV Name                /dev/cinder-volumes/volume-00000001
VG Name                cinder-volumes
LV UUID                gI8hta-p21U-IW2q-hRN1-nTzN-UC2G-dKbdKr
LV Write Access        read/write
LV snapshot status     source of
                       /dev/cinder-volumes/volume-00000026-snap
 [active]
LV Status              available
# open                 1
LV Size                15,00 GiB
Current LE             3840
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           251:13

--- Logical volume ---
LV Name                /dev/cinder-volumes/volume-00000001-snap
VG Name                cinder-volumes
LV UUID                HlW3Ep-g5I8-KGQb-IRvi-IRYU-lIKe-wE9zYr
LV Write Access        read/write
LV snapshot status     active destination for /dev/cinder-volumes/
volume-00000026
LV Status              available
# open                 0
LV Size                15,00 GiB
Current LE             3840
COW-table size         10,00 GiB
COW-table LE           2560
Allocated to snapshot  0,00%
Snapshot chunk size    4,00 KiB
Segments               1
Allocation             inherit
Read ahead sectors     auto
- currently set to     256
Block device           251:14
```

2. **Partition table discovery**

   a. To exploit the snapshot with the **tar** command, mount your partition on the Block Storage service server.

      The **kpartx** utility discovers and maps table partitions. You can use it to view partitions that are created inside the instance. Without using the partitions created inside instances, you cannot see its content and create efficient backups.

      ```
      # kpartx -av /dev/cinder-volumes/volume-00000001-snapshot
      ```

      Note

      On a Debian-based distribution, you can use the **apt-get install kpartx** command to install **kpartx**.

      If the tools successfully find and map the partition table, no errors are returned.

   b. To check the partition table map, run this command:

      ```
      $ ls /dev/mapper/nova*
      ```

      You can see the `cinder--volumes-volume--00000001--snapshot1` partition.

      If you created more than one partition on that volume, you see several partitions; for example: `cinder--volumes-volume--00000001--snapshot2`, `cinder--volumes-volume--00000001--snapshot3`, and so on.

   c. Mount your partition:

      ```
      # mount /dev/mapper/cinder--volumes-volume--volume--00000001--
      snapshot1 /mnt
      ```

      If the partition mounts successfully, no errors are returned.

      You can directly access the data inside the instance. If a message prompts you for a partition or you cannot mount it, determine whether enough space was allocated for the snapshot or the **kpartx** command failed to discover the partition table.

      Allocate more space to the snapshot and try the process again.

3. **Use the tar command to create archives**

   Create a backup of the volume:

   ```
   $ tar --exclude="lost+found" --exclude="some/data/to/exclude" -czf
    volume-00000001.tar.gz -C /mnt/ /backup/destination
   ```

   This command creates a `tar.gz` file that contains the data, *and data only*. This ensures that you do not waste space by backing up empty sectors.

4. **Checksum calculation I**

   You should always have the checksum for your backup files. When you transfer the same file over the network, you can run a checksum calculation to ensure that your file was not corrupted during its transfer. The checksum is a unique ID for a file. If the checksums are different, the file is corrupted.

   Run this command to run a checksum for your file and save the result to a file:

   ```
   $ sha1sum volume-00000001.tar.gz > volume-00000001.checksum
   ```

   Note

   Use the **sha1sum** command carefully because the time it takes to complete the calculation is directly proportional to the size of the file.

   For files larger than around 4 to 6 GB, and depending on your CPU, the process might take a long time.

5. **After work cleaning**

   Now that you have an efficient and consistent backup, use this command to clean up the file system:

   a. Unmount the volume:

      ```
      unmount /mnt
      ```

   b. Delete the partition table:

      ```
      kpartx -dv /dev/cinder-volumes/volume-00000001-snapshot
      ```

   c. Remove the snapshot:

      ```
      lvremove -f /dev/cinder-volumes/volume-00000001-snapshot
      ```

   Repeat these steps for all your volumes.

6. **Automate your backups**

   Because more and more volumes might be allocated to your Block Storage service, you might want to automate your backups. The SCR_5005_V01_NUAC-OPENSTACK-EBS-volumes-backup.sh script assists you with this task. The script performs the operations from the previous example, but also provides a mail report and runs the backup based on the `backups_retention_days` setting.

   Launch this script from the server that runs the Block Storage service.

   This example shows a mail report:

```
Backup Start Time - 07/10 at 01:00:01
Current retention - 7 days

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-00000019/
volume-00000019_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-00000019 - 0 h 1 m and 21 seconds. Size - 3,
5G

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-0000001a/
volume-0000001a_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-0000001a - 0 h 4 m and 15 seconds. Size - 6,
9G
---------------------------------------
Total backups size - 267G - Used space : 35%
Total execution time - 1 h 75 m and 35 seconds
```

The script also enables you to SSH to your instances and run a **mysqldump** command into them. To make this work, enable the connection to the Compute project keys. If you do not want to run the **mysqldump** command, you can add `enable_mysql_dump=0` to the script to turn off this functionality.

Migrate volumes

OpenStack has the ability to migrate volumes between back-ends. Migrating a volume transparently moves its data from the current back-end for the volume to a new one. This is an administrator function, and can be used for functions including storage evacuation (for maintenance or decommissioning), or manual optimizations (for example, performance, reliability, or cost).

These workflows are possible for a migration:

1. If the storage can migrate the volume on its own, it is given the opportunity to do so. This allows the Block Storage driver to enable optimizations that the storage might be able to perform. If the back-end is not able to perform the migration, the Block Storage uses one of two generic flows, as follows.

2. If the volume is not attached, the Block Storage service creates a volume and copies the data from the original to the new volume.

   Note

   While most back-ends support this function, not all do. See the driver documentation in the *OpenStack Configuration Reference* for more details.

3. If the volume is attached to a VM instance, the Block Storage creates a volume, and calls Compute to copy the data from the original to the new volume. Currently this is supported only by the Compute libvirt driver.

As an example, this scenario shows two LVM back-ends and migrates an attached volume from one to the other. This scenario uses the third migration flow.

First, list the available back-ends:

```
# cinder-manage host list
server1@lvmstorage-1    zone1
server2@lvmstorage-2    zone1
```

Next, as the admin user, you can see the current status of the volume (replace the example ID with your own):

```
$ cinder show 6088f80a-f116-4331-ad48-9afb0dfb196c
+--------------------------------+--------------------------------------+
|            Property            |                Value                 |
+--------------------------------+--------------------------------------+
|          attachments           |                [...]                 |
|       availability_zone        |                zone1                 |
|            bootable            |                False                 |
|           created_at           |      2013-09-01T14:53:22.000000      |
|      display_description       |                 test                 |
|          display_name          |                 test                 |
|               id               | 6088f80a-f116-4331-ad48-9afb0dfb196c |
|            metadata            |                  {}                  |
|     os-vol-host-attr:host      |         server1@lvmstorage-1         |
| os-vol-mig-status-attr:migstat |                 None                 |
| os-vol-mig-status-attr:name_id |                 None                 |
|  os-vol-tenant-attr:tenant_id  |   6bdd8f41203e4149b5d559769307365e   |
|              size              |                  2                   |
|          snapshot_id           |                 None                 |
|          source_volid          |                 None                 |
|             status             |                in-use                |
|          volume_type           |                 None                 |
+--------------------------------+--------------------------------------+
```

Note these attributes:

- `os-vol-host-attr:host` - the volume's current back-end.
- `os-vol-mig-status-attr:migstat` - the status of this volume's migration (None means that a migration is not currently in progress).
- `os-vol-mig-status-attr:name_id` - the volume ID that this volume's name on the back-end is based on. Before a volume is ever migrated, its name on the back-end storage may be based on the volume's ID (see the `volume_name_template` configuration parameter). For example, if `volume_name_template` is kept as the default value (`volume-%s`), your first LVM back-end has a logical volume named `volume-6088f80a-f116-4331-ad48-9afb0dfb196c`. During the course of a migration, if you create a volume and copy over the data, the volume get the new name but keeps its original ID. This is exposed by the `name_id` attribute.

Note

If you plan to decommission a block storage node, you must stop the `cinder` volume service on the node after performing the migration.

On nodes that run CentOS, Fedora, openSUSE, RedHat Enterprise Linux, or SUSE Linux Enterprise, run:

```
# service openstack-cinder-volume stop
# chkconfig openstack-cinder-volume off
```

On nodes that run Ubuntu or Debian, run:

```
# service cinder-volume stop
# chkconfig cinder-volume off
```

Stopping the `cinder` volume service will prevent volumes from being allocated to the node.

Migrate this volume to the second LVM back-end:

```
$ cinder migrate 6088f80a-f116-4331-ad48-9afb0dfb196c server2@lvmstorage-2
```

You can use the **cinder show** command to see the status of the migration. While migrating, the `migstat` attribute shows states such as `migrating` or `completing`. On error, `migstat` is set to None and the `host` attribute shows the original host. On success, in this example, the output looks like:

```
+--------------------------------+--------------------------------------+
|            Property            |                Value                 |
+--------------------------------+--------------------------------------+
|          attachments           |                [...]                 |
|       availability_zone        |                zone1                 |
|            bootable            |                False                 |
|           created_at           |      2013-09-01T14:53:22.000000      |
|      display_description       |                 test                 |
|          display_name          |                 test                 |
|               id               | 6088f80a-f116-4331-ad48-9afb0dfb196c |
|            metadata            |                  {}                  |
|     os-vol-host-attr:host      |         server2@lvmstorage-2         |
| os-vol-mig-status-attr:migstat |                 None                 |
| os-vol-mig-status-attr:name_id | 133d1f56-9ffc-4f57-8798-d5217d851862 |
|  os-vol-tenant-attr:tenant_id  |   6bdd8f41203e4149b5d559769307365e   |
|              size              |                  2                   |
|          snapshot_id           |                 None                 |
|          source_volid          |                 None                 |
|             status             |                in-use                |
|          volume_type           |                 None                 |
+--------------------------------+--------------------------------------+
```

Note that `migstat` is None, `host` is the new host, and `name_id` holds the ID of the volume created by the migration. If you look at the second LVM back end, you find the logical volume `volume-133d1f56-9ffc-4f57-8798-d5217d851862`.

Note

The migration is not visible to non-admin users (for example, through the volume `status`). However, some operations are not allowed while a migration is taking place, such as attaching/detaching a volume and deleting a volume. If a user performs such an action during a migration, an error is returned.

Note

Migrating volumes that have snapshots are currently not allowed.

Gracefully remove a GlusterFS volume from usage

Configuring the `cinder` volume service to use GlusterFS involves creating a shares file (for example, `/etc/cinder/glusterfs`). This shares file lists each GlusterFS volume (with its corresponding storage server) that the `cinder` volume service can use for back end storage.

To remove a GlusterFS volume from usage as a back end, delete the volume's corresponding entry from the shares file. After doing so, restart the Block Storage services.

To restart the Block Storage services on CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, or SUSE Linux Enterprise, run:

```
# for i in api scheduler volume; do service openstack-cinder-$i restart; done
```

To restart the Block Storage services on Ubuntu or Debian, run:

```
# for i in api scheduler volume; do service cinder-${i} restart; done
```

Restarting the Block Storage services will prevent the `cinder` volume service from exporting the deleted GlusterFS volume. This will prevent any instances from mounting the volume from that point onwards.

However, the removed GlusterFS volume might still be mounted on an instance at this point. Typically, this is the case when the volume was already mounted while its entry was deleted from the shares file. Whenever this occurs, you will have to unmount the volume as normal after the Block Storage services are restarted.

Back up and restore volumes

The **cinder** command-line interface provides the tools for creating a volume backup. You can restore a volume from a backup as long as the backup's associated database information (or backup metadata) is intact in the Block Storage database.

Run this command to create a backup of a volume:

```
$ cinder backup-create VOLUME
```

Where *`VOLUME`* is the name or ID of the volume.

The previous command will also return a backup ID. Use this backup ID when restoring the volume, as in:

```
$ cinder backup-restore backup_ID
```

As mentioned earlier, volume backups are dependent on the Block Storage database. Because of this, we recommend that you also back up your Block Storage database regularly in order to ensure data recovery.

Note

Alternatively, you can export and save the metadata of selected volume backups. Doing so will preclude the need to back up the entire Block Storage database. This is particularly useful if you only need a small subset of volumes to survive a catastrophic database failure.

For more information on how to export and import volume backup metadata, see the section called “Export and import backup metadata” [158].

Export and import backup metadata

A volume backup can only be restored on the same Block Storage service. This is because restoring a volume from a backup requires metadata available on the database used by the Block Storage service.

Note

For information on how to back up and restore a volume, see the section called “Back up and restore volumes” [157].

You can, however, export the metadata of a volume backup. To do so, run this command as an OpenStack `admin` user (presumably, after creating a volume backup):

```
$ cinder backup-export backup_ID
```

Where *backup_ID* is the volume backup's ID. This command should return the backup's corresponding database information as encoded string metadata.

Exporting and storing this encoded string metadata allows you to completely restore the backup, even in the event of a catastrophic database failure. This will preclude the need to back up the entire Block Storage database, particularly if you only need to keep complete backups of a small subset of volumes.

In addition, having a volume backup and its backup metadata also provides volume portability. Specifically, backing up a volume and exporting its metadata will allow you to restore the volume on a completely different Block Storage database, or even on a different cloud service. To do so, first import the backup metadata to the Block Storage database and then restore the backup.

To import backup metadata, run the following command as an OpenStack `admin`:

```
$ cinder backup-import metadata
```

Where *metadata* is the backup metadata exported earlier.

Once you have imported the backup metadata into a Block Storage database, restore the volume (the section called “Back up and restore volumes” [157]).

Troubleshoot your installation

This section provides useful tips to help troubleshoot your Block Storage installation.

Troubleshoot the Block Storage configuration

Most Block Storage errors are caused by incorrect volume configurations that result in volume creation failues. To resolve these failures, review these logs:

- `cinder-api` log (`/var/log/cinder/api.log`)
- `cinder-volume` log (`/var/log/cinder/volume.log`)

The `cinder-api` log is useful for determining if you have endpoint or connectivity issues. If you send a request to create a volume and it fails, review the `cinder-api` log to determine whether the request made it to the Block Storage service. If the request is logged and you see no errors or trace-backs, check the `cinder-volume` log for errors or trace-backs.

Note

Create commands are listed in the `cinder-api` log.

These entries in the `cinder.openstack.common.log` file can be used to assist in troubleshooting your block storage configuration.

```
# Print debugging output (set logging level to DEBUG instead
# of default WARNING level). (boolean value)
#debug=false

# Print more verbose output (set logging level to INFO instead
# of default WARNING level). (boolean value)
#verbose=false

# Log output to standard error (boolean value)
#use_stderr=true

# Default file mode used when creating log files (string
# value)
#logfile_mode=0644

# format string to use for log messages with context (string
# value)
#logging_context_format_string=%(asctime)s.%(msecs)03d %(levelname)s %(name)s
 [%(request_id)s %(user)s %(tenant)s] %(instance)s%(message)s

# format string to use for log mes #logging_default_format_string=%(asctime)s.
%(msecs)03d %(process)d %(levelname)s %(name)s [-] %(instance)s%(message)s

# data to append to log format when level is DEBUG (string
# value)
#logging_debug_format_suffix=%(funcName)s %(pathname)s:%(lineno)d

# prefix each line of exception output with this format
# (string value)
#logging_exception_prefix=%(asctime)s.%(msecs)03d %(process)d TRACE %(name)s
 %(instance)s

# list of logger=LEVEL pairs (list value)
#default_log_levels=amqplib=WARN,sqlalchemy=WARN,boto=WARN,suds=INFO,keystone=
INFO,eventlet.wsgi.server=WARNsages without context
# (string value)

# If an instance is passed with the log message, format it
# like this (string value)
#instance_format="[instance: %(uuid)s]"

# If an instance UUID is passed with the log message, format
# it like this (string value)
# A logging.Formatter log message format string which may use
# any of the available logging.LogRecord attributes. Default:
```

```
# %(default)s (string value)
#log_format=%(asctime)s %(levelname)8s [%(name)s] %(message)s

# Format string for %%(asctime)s in log records. Default:
# %(default)s (string value)
#log_date_format=%Y-%m-%d %H:%M:%S

# (Optional) Name of log file to output to. If not set,
# logging will go to stdout. (string value)
#log_file=<None>

# (Optional) The directory to keep log files in (will be
# prepended to --log-file) (string value)
#log_dir=<None>
#instance_uuid_format="[instance: %(uuid)s]"

# If this option is specified, the logging configuration file
# specified is used and overrides any other logging options
# specified. Please see the Python logging module
# documentation for details on logging configuration files.
# (string value) # Use syslog for logging. (boolean value)
#use_syslog=false

# syslog facility to receive log lines (string value)
#syslog_log_facility=LOG_USER
#log_config=<None>
```

These common issues might occur during configuration. To correct, use these suggested solutions.

- Issues with `state_path` and `volumes_dir` settings.

  The OpenStack Block Storage uses **tgtd** as the default iscsi helper and implements persistent targets. This means that in the case of a tgt restart or even a node reboot your existing volumes on that node will be restored automatically with their original IQN.

  In order to make this possible the iSCSI target information needs to be stored in a file on creation that can be queried in case of restart of the tgt daemon. By default, Block Storage uses a `state_path` variable, which if installing with Yum or APT should be set to `/var/lib/cinder/`. The next part is the `volumes_dir` variable, by default this just simply appends a "`volumes`" directory to the `state_path`. The result is a file-tree `/var/lib/cinder/volumes/`.

  While this should all be handled by the installer, it can go wrong. If you have trouble creating volumes and this directory does not exist you should see an error message in the `cinder-volume` log indicating that the `volumes_dir` does not exist, and it should provide information about which path it was looking for.

- The persistent tgt include file.

  Along with the `volumes_dir` option, the iSCSI target driver also needs to be configured to look in the correct place for the persist files. This is a simple entry in the `/etc/tgt/conf.d` file that you should have set when you installed OpenStack. If issues occur, verify that you have a `/etc/tgt/conf.d/cinder.conf` file.

  If the file is not present, create it with this command:

```
# echo 'include /var/lib/cinder/volumes/ *' >> /etc/tgt/conf.d/cinder.conf
```

- No sign of attach call in the `cinder-api` log.

  This is most likely going to be a minor adjustment to your `nova.conf` file. Make sure that your `nova.conf` has this entry:

  ```
  volume_api_class=nova.volume.cinder.API
  ```

- Failed to create iscsi target error in the `cinder-volume.log` file.

  ```
  2013-03-12 01:35:43 1248 TRACE cinder.openstack.common.rpc.amqp
   ISCSITargetCreateFailed: Failed to create iscsi target for volume
   volume-137641b2-af72-4a2f-b243-65fdccd38780.
  ```

  You might see this error in `cinder-volume.log` after trying to create a volume that is 1 GB. To fix this issue:

  Change content of the `/etc/tgt/targets.conf` from `include /etc/tgt/conf.d/*.conf` to `include /etc/tgt/conf.d/cinder_tgt.conf`, as follows:

  ```
  include /etc/tgt/conf.d/cinder_tgt.conf
  include /etc/tgt/conf.d/cinder.conf
  default-driver iscsi
  ```

  Restart `tgt` and `cinder-*` services so they pick up the new configuration.

Multipath Call Failed Exit

Problem

Multipath call failed exit. This warning occurs in the Compute log if you do not have the optional `multipath-tools` package installed on the compute node. This is an optional package and the volume attachment does work without the multipath tools installed. If the `multipath-tools` package is installed on the compute node, it is used to perform the volume attachment. The IDs in your message are unique to your system.

```
WARNING nova.storage.linuxscsi [req-cac861e3-8b29-4143-8f1b-705d0084e571 admin
            admin|req-cac861e3-8b29-4143-8f1b-705d0084e571 admin admin]
 Multipath call failed exit
            (96)
```

Solution

Run the following command on the compute node to install the `multipath-tools` packages.

```
# apt-get install multipath-tools
```

Failed to Attach Volume, Missing sg_scan

Problem

Failed to attach volume to an instance, `sg_scan` file not found. This warning and error occur when the sg3-utils package is not installed on the compute node. The IDs in your message are unique to your system:

```
ERROR nova.compute.manager [req-cf2679fd-dd9e-4909-807f-48fe9bda3642 admin
 admin|req-cf2679fd-dd9e-4909-807f-48fe9bda3642 admin admin]
[instance: 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5|instance:
 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5]
Failed to attach volume  4cc104c4-ac92-4bd6-9b95-c6686746414a at /dev/vdcTRACE
 nova.compute.manager
[instance:  7d7c92e0-49fa-4a8e-87c7-73f22a9585d5|instance:
 7d7c92e0-49fa-4a8e-87c7-73f22a9585d5]
Stdout: '/usr/local/bin/nova-rootwrap: Executable not found: /usr/bin/sg_scan
```

Solution

Run this command on the compute node to install the sg3-utils package:

```
# apt-get install sg3-utils
```

Failed to attach volume after detaching

Problem

These errors appear in the `cinder-volume.log` file.

```
2013-05-03 15:16:33 INFO [cinder.volume.manager] Updating volume status
2013-05-03 15:16:33 DEBUG [hp3parclient.http]
REQ: curl -i https://10.10.22.241:8080/api/v1/cpgs -X GET -H "X-Hp3Par-Wsapi-Sessionkey:
 48dc-b69ed2e5
f259c58e26df9a4c85df110c-8d1e8451" -H "Accept: application/json" -H "User-Agent:
 python-3parclient"

2013-05-03 15:16:33 DEBUG [hp3parclient.http] RESP:{'content-length': 311, 'content-
type': 'text/plain',
'status': '400'}

2013-05-03 15:16:33 DEBUG [hp3parclient.http] RESP BODY:Second simultaneous read on
 fileno 13 detected.
Unless you really know what you're doing, make sure that only one greenthread can read
 any particular socket.
Consider using a pools.Pool. If you do know what you're doing and want to disable this
 error,
call eventlet.debug.hub_multiple_reader_prevention(False)

2013-05-03 15:16:33 ERROR [cinder.manager] Error during VolumeManager.
_report_driver_status: Bad request (HTTP 400)
Traceback (most recent call last):
File "/usr/lib/python2.7/dist-packages/cinder/manager.py", line 167, in periodic_tasks
 task(self, context)
File "/usr/lib/python2.7/dist-packages/cinder/volume/manager.py", line 690, in
 _report_driver_status volume_stats =
self.driver.get_volume_stats(refresh=True)
File "/usr/lib/python2.7/dist-packages/cinder/volume/drivers/san/hp/hp_3par_fc.py", line
 77, in get_volume_stats stats =
self.common.get_volume_stats(refresh, self.client)
File "/usr/lib/python2.7/dist-packages/cinder/volume/drivers/san/hp/hp_3par_common.py",
 line 421, in get_volume_stats cpg =
client.getCPG(self.config.hp3par_cpg)
File "/usr/lib/python2.7/dist-packages/hp3parclient/client.py", line 231, in getCPG cpgs
 = self.getCPGs()
File "/usr/lib/python2.7/dist-packages/hp3parclient/client.py", line 217, in getCPGs
 response, body = self.http.get('/cpgs')
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 255, in get return
 self._cs_request(url, 'GET', **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 224, in _cs_request
 **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 198, in _time_request
 resp, body = self.request(url, method, **kwargs)
File "/usr/lib/python2.7/dist-packages/hp3parclient/http.py", line 192, in request raise
 exceptions.from_response(resp, body)
HTTPBadRequest: Bad request (HTTP 400)
```

Solution

You need to update your copy of the `hp_3par_fc.py` driver which contains the synchronization code.

Duplicate 3PAR host

Problem

This error may be caused by a volume being exported outside of OpenStack using a host name different from the system name that OpenStack expects. This error could be displayed with the IQN if the host was exported using iSCSI.

```
Duplicate3PARHost: 3PAR Host already exists: Host wwn 50014380242B9750
 already used by host cld4b5ubuntuW(id = 68. The hostname must be called
 'cld4b5ubuntu‘.
```

Solution

Change the 3PAR host name to match the one that OpenStack expects. The 3PAR host constructed by the driver uses just the local hostname, not the fully qualified domain name (FQDN) of the compute host. For example, if the FQDN was *myhost.example.com*, just *myhost* would be used as the 3PAR hostname. IP addresses are not allowed as host names on the 3PAR storage server.

Failed to attach volume after detaching

Problem

Failed to attach a volume after detaching the same volume.

Solution

You must change the device name on the **nova-attach** command. The VM might not clean up after a **nova-detach** command runs. This example shows how the **nova-attach** command fails when you use the `vdb`, `vdc`, or `vdd` device names:

```
# ls -al /dev/disk/by-path/
total 0
drwxr-xr-x 2 root root 200 2012-08-29 17:33 .
drwxr-xr-x 5 root root 100 2012-08-29 17:33 ..
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-virtio0
 -> ../../vda
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-
virtio0-part1 -> ../../vda1
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-
virtio0-part2 -> ../../vda2
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:04.0-virtio-pci-
virtio0-part5 -> ../../vda5
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:06.0-virtio-pci-virtio2
 -> ../../vdb
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:08.0-virtio-pci-virtio3
 -> ../../vdc
lrwxrwxrwx 1 root root 9 2012-08-29 17:33 pci-0000:00:09.0-virtio-pci-virtio4
 -> ../../vdd
lrwxrwxrwx 1 root root 10 2012-08-29 17:33 pci-0000:00:09.0-virtio-pci-
virtio4-part1 -> ../../vdd1
```

You might also have this problem after attaching and detaching the same volume from the same VM with the same mount point multiple times. In this case, restart the KVM host.

Failed to attach volume, systool is not installed

Problem

This warning and error occurs if you do not have the required `sysfsutils` package installed on the compute node.

```
WARNING nova.virt.libvirt.utils [req-1200f887-c82b-4e7c-a891-fac2e3735dbb
 admin admin|req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin] systool is
 not installed
ERROR nova.compute.manager [req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin
 admin|req-1200f887-c82b-4e7c-a891-fac2e3735dbb admin admin]
[instance: df834b5a-8c3f-477a-be9b-47c97626555c|instance: df834b5a-8c3f-477a-
be9b-47c97626555c]
Failed to attach volume 13d5c633-903a-4764-a5a0-3336945b1db1 at /dev/vdk.
```

Solution

Run the following command on the compute node to install the `sysfsutils` packages.

```
# apt-get install sysfsutils
```

Failed to connect volume in FC SAN

Problem

Compute node failed to connect to a volume in a Fibre Channel (FC) SAN configuration. The WWN may not be zoned correctly in your FC SAN that links the compute host to the storage array.

```
ERROR nova.compute.manager [req-2ddd5297-e405-44ab-aed3-152cd2cfb8c2 admin
 demo|req-2ddd5297-e405-44ab-aed3-152cd2cfb8c2 admin demo] [instance:
 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3|instance: 60ebd6c7-c1e3-4bf0-8ef0-
f07aa4c3d5f3]
Failed to connect to volume 6f6a6a9c-dfcf-4c8d-b1a8-4445ff883200 while
 attaching at /dev/vdjTRACE nova.compute.manager [instance: 60ebd6c7-
c1e3-4bf0-8ef0-f07aa4c3d5f3|instance: 60ebd6c7-c1e3-4bf0-8ef0-f07aa4c3d5f3]
Traceback (most recent call last):…f07aa4c3d5f3\] ClientException: The server
 has either erred or is incapable of performing the requested operation.(HTTP
 500)(Request-ID: req-71e5132b-21aa-46ee-b3cc-19b5b4ab2f00)
```

Solution

The network administrator must configure the FC SAN fabric by correctly zoning the WWN (port names) from your compute node HBAs.

Cannot find suitable emulator for x86_64

Problem

When you attempt to create a VM, the error shows the VM is in the `BUILD` then `ERROR` state.

Solution

On the KVM host run, `cat /proc/cpuinfo`. Make sure the `vme` and `svm` flags are set.

Follow the instructions in the enabling KVM section of the *Configuration Reference* to enable hardware virtualization support in your BIOS.

Non-existent host

Problem

This error could be caused by a volume being exported outside of OpenStack using a host name different from the system name that OpenStack expects. This error could be displayed with the IQN if the host was exported using iSCSI.

```
2013-04-19 04:02:02.336 2814 ERROR cinder.openstack.common.rpc.common [-]
 Returning exception Not found (HTTP 404)
NON_EXISTENT_HOST - HOST '10' was not found to caller.
```

Solution

Host names constructed by the driver use just the local hostname, not the fully qualified domain name (FQDN) of the Compute host. For example, if the FQDN was *myhost.example.com*, just *myhost* would be used as the 3PAR hostname. IP addresses are not allowed as host names on the 3PAR storage server.

Non-existent VLUN

Problem

This error occurs if the 3PAR host exists with the correct host name that the OpenStack Block Storage drivers expect but the volume was created in a different Domain.

```
HTTPNotFound: Not found (HTTP 404) NON_EXISTENT_VLUN - VLUN 'osv-
DqT7CE3mSrWi4gZJmHAP-Q' was not found.
```

Solution

The `hp3par_domain` configuration items either need to be updated to use the domain the 3PAR host currently resides in, or the 3PAR host needs to be moved to the domain that the volume was created in.

7. Networking

Table of Contents

Learn OpenStack Networking concepts, architecture, and basic and advanced **neutron** and **nova** command-line interface (CLI) commands.

Introduction to Networking

The Networking service, code-named Neutron, provides an API that lets you define network connectivity and addressing in the cloud. The Networking service enables operators to leverage different networking technologies to power their cloud networking. The Networking service also provides an API to configure and manage a variety of network services ranging from L3 forwarding and NAT to load balancing, edge firewalls, and IPSEC VPN.

For a detailed description of the Networking API abstractions and their attributes, see the *OpenStack Networking API v2.0 Reference*.

Networking API

Networking is a virtual network service that provides a powerful API to define the network connectivity and IP addressing that devices from other services, such as Compute, use.

The Compute API has a virtual server abstraction to describe computing resources. Similarly, the Networking API has virtual network, subnet, and port abstractions to describe networking resources.

Table 7.1. Networking resources

Resource	Description
Network	An isolated L2 segment, analogous to VLAN in the physical networking world.
Subnet	A block of v4 or v6 IP addresses and associated configuration state.
Port	A connection point for attaching a single device, such as the NIC of a virtual server, to a virtual network. Also describes the associated network configuration, such as the MAC and IP addresses to be used on that port.

You can configure rich network topologies by creating and configuring networks and subnets, and then instructing other OpenStack services like Compute to attach virtual devices to ports on these networks.

In particular, Networking supports each tenant having multiple private networks, and allows tenants to choose their own IP addressing scheme (even if those IP addresses overlap with those that other tenants use). The Networking service:

- Enables advanced cloud networking use cases, such as building multi-tiered web applications and enabling migration of applications to the cloud without changing IP addresses.
- Offers flexibility for the cloud administrator to customize network offerings.
- Enables developers to extend the Networking API. Over time, the extended functionality becomes part of the core Networking API.

Configure SSL support for networking API

OpenStack Networking supports SSL for the Networking API server. By default, SSL is disabled but you can enable it in the `neutron.conf` file.

Set these options to configure SSL:

`use_ssl = True`	Enables SSL on the networking API server.
`ssl_cert_file = /path/to/certfile`	Certificate file that is used when you securely start the Networking API server.
`ssl_key_file = /path/to/keyfile`	Private key file that is used when you securely start the Networking API server.
`ssl_ca_file = /path/to/cafile`	Optional. CA certificate file that is used when you securely start the Networking API server. This file verifies connecting clients. Set this option when API clients must authenticate to the API server by using SSL certificates that are signed by a trusted CA.
`tcp_keepidle = 600`	The value of TCP_KEEPIDLE, in seconds, for each server socket when starting the API server. Not supported on OS X.
`retry_until_window = 30`	Number of seconds to keep retrying to listen.
`backlog = 4096`	Number of backlog requests with with to configure the socket.

Load Balancing-as-a-Service (LBaaS) overview

Load Balancing-as-a-Service (LBaaS) enables Networking to distribute incoming requests evenly between designated instances. This ensures the workload is shared predictably among instances, and allows more effective use of system resources. Incoming requests are distributed using one of these load balancing methods:

Round robin	Rotates requests evenly between multiple instances.
Source IP	Requests from a unique source IP address are consistently directed to the same instance.
Least connections	Allocates requests to the instance with the least number of active connections.

Table 7.2. LBaaS features

Feature	Description
Monitors	LBaaS provides availability monitoring with the **ping**, TCP, HTTP and HTTPS GET methods. *Monitors* are implemented to determine whether pool members are available to handle requests.
Management	LBaaS is managed using a variety of tool sets. The REST `API` is available for programmatic administration and scripting. Users perform administrative management of load balancers through either the CLI (**neutron**) or the OpenStack dashboard.
Connection limits	Ingress traffic can be shaped with *connection limits*. This feature allows workload control, and can also assist with mitigating DoS (Denial of Service) attacks.
Session persistence	LBaaS supports session persistence by ensuring incoming requests are routed to the same instance within a pool of multiple instances. LBaaS supports routing decisions based on cookies and source IP address.

Firewall-as-a-Service (FWaaS) overview

The *Firewall-as-a-Service (FWaaS)* plug-in adds perimeter firewall management to Networking. FWaaS uses iptables to apply firewall policy to all Networking routers within a project. FWaaS supports one firewall policy and logical firewall instance per project.

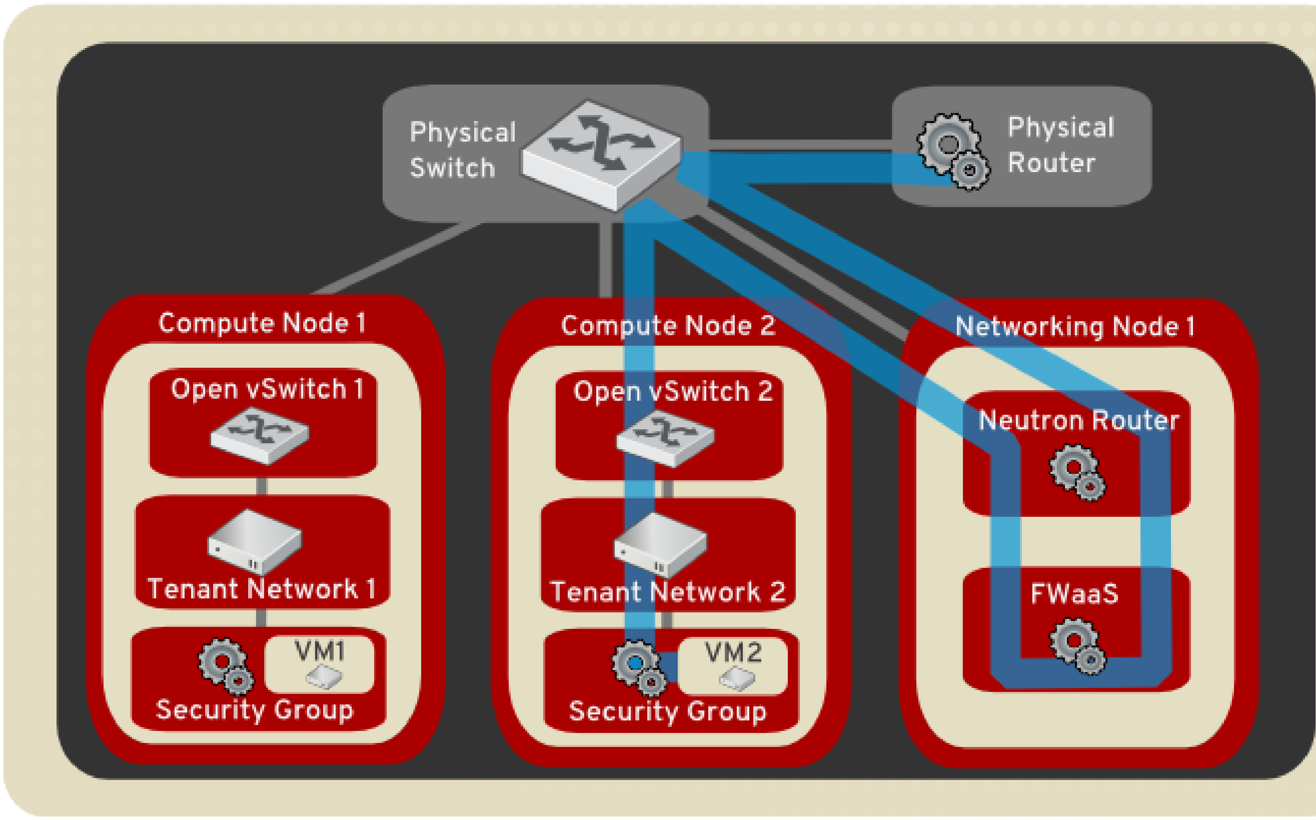
Whereas security groups operate at the instance-level, FWaaS operates at the perimeter by filtering traffic at the neutron router.

Note

FWaaS is currently in technical preview; untested operation is not recommended.

The example diagram below illustrates the flow of ingress and egress traffic for the VM2 instance:

Figure 7.1. FWaaS architecture



Enable FWaaS. Enable the FWaaS plugin in the `neutron.conf` file:

```
service_plugins = neutron.services.firewall.fwaas_plugin.FirewallPlugin
[service_providers]
service_provider=LOADBALANCER:Haproxy:neutron.services.loadbalancer.drivers.
haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
[fwaas]
driver = neutron.services.firewall.drivers.linux.iptables_fwaas.
IptablesFwaasDriver
enabled = True
```

FWaaS management options are available in OpenStack dashboard. Enable the option in the file typically located on the controller node: `/usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py`

```
'enable_firewall' = True
```

Procedure 7.1. Configure Firewall-as-a-Service

First create the firewall rules and create a policy that contains them, then create a firewall that applies the policy:

1. Create a firewall rule:

   ```
   $ neutron firewall-rule-create --protocol <tcp|udp|icmp|any> --
   destination-port <port-range> --action <allow|deny>
   ```

The CLI requires a protocol value; if the rule is protocol agnostic, the 'any' value can be used.

2. Create a firewall policy:

```
$ neutron firewall-policy-create --firewall-rules  "<firewall-rule IDs or
 names separated by space>" myfirewallpolicy
```

The order of the rules specified above is important.You can create a firewall policy without and rules and add rules later either with the update operation (when adding multiple rules) or with the insert-rule operations (when adding a single rule). Please check the CLI Reference for more details on these operations.

Note

FWaaS always adds a default `deny all` rule at the lowest precedence of each policy. Consequently, a firewall policy with no rules blocks all traffic by default.

3. Create a firewall:

```
$ neutron firewall-create  <firewall-policy-uuid>
```

Note

The firewall remains in **PENDING_CREATE** state until a Networking router is created, and an interface is attached.

Allowed-address-pairs.

`Allowed-address-pairs` allow you to specify mac_address/ip_address(cidr) pairs that pass through a port regardless of subnet. This enables the use of protocols such as VRRP, which floats an IP address between two instances to enable fast data plane failover.

Note

The allowed-address-pairs extension is currently only supported by these plug-ins: ML2, Open vSwitch, and VMware NSX.

Basic allowed-address-pairs operations.

- Create a port with a specific allowed-address-pairs:

```
$ neutron port-create net1 --allowed-address-pairs type=dict list=true
 mac_address=<mac_address>,ip_address=<ip_cidr>
```

- Update a port adding allowed-address-pairs:

```
$ neutron port-update  <port-uuid> --allowed-address-pairs type=dict list=
true mac_address=<mac_address>,ip_address=<ip_cidr>
```

Note

OpenStack Networking prevents setting an allowed-address-pair that matches the mac_address and ip_address of a port. This is because that would have no effect since traffic matching the mac_address and ip_address is already allowed to pass through the port.

Plug-in architecture

The original Compute network implementation assumed a basic model of isolation through Linux VLANs and IP tables. Networking introduces support for vendor *plug-ins*, which offer a custom back-end implementation of the Networking API. A plug-in can use a variety of technologies to implement the logical API requests. Some Networking plug-ins might use basic Linux VLANs and IP tables, while others might use more advanced technologies, such as L2-in-L3 tunneling or OpenFlow, to provide similar benefits.

Table 7.3. Available networking plug-ins

Plug-in	Documentation
Big Switch Plug-in (Floodlight REST Proxy)	This guide and http://www.openflowhub.org/display/floodlightcontroller/Neutron+REST+Proxy+Plugin
Brocade Plug-in	This guide and https://wiki.openstack.org/wiki/Brocade-neutron-plugin
Cisco	http://wiki.openstack.org/cisco-neutron
Cloudbase Hyper-V Plug-in	http://www.cloudbase.it/quantum-hyper-v-plugin/
Linux Bridge Plug-in	http://wiki.openstack.org/Neutron-Linux-Bridge-Plugin
Mellanox Plug-in	https://wiki.openstack.org/wiki/Mellanox-Neutron/
Midonet Plug-in	http://www.midokura.com/
ML2 (Modular Layer 2) Plug-in	https://wiki.openstack.org/wiki/Neutron/ML2
NEC OpenFlow Plug-in	https://wiki.openstack.org/wiki/Neutron/NEC_OpenFlow_Plugin
Open vSwitch Plug-in	This guide.
PLUMgrid	This guide and https://https://wiki.openstack.org/wiki/PLUMgrid-Neutron
Ryu Plug-in	This guide and https://github.com/osrg/ryu/wiki/OpenStack
VMware NSX Plug-in	This guide and NSX Product Overview, NSX Product Support

Plug-ins can have different properties for hardware requirements, features, performance, scale, or operator tools. Because Networking supports a large number of plug-ins, the cloud administrator can weigh options to decide on the right networking technology for the deployment.

In the Havana release, OpenStack Networking introduces the *Modular Layer 2 (ML2) plug-in* that enables the use of multiple concurrent mechanism drivers. This capability aligns with the complex requirements typically found in large heterogeneous environments. It currently works with the existing Open vSwitch, Linux Bridge, and Hyper-v L2 agents. The ML2 framework simplifies the addition of support for new L2 technologies and reduces the effort that is required to add and maintain them compared to earlier large plug-ins.

Plug-in deprecation notice

The Open vSwitch and Linux Bridge plug-ins are deprecated in the Havana release and will be removed in the Icehouse release. The features in these plug-ins are now part of the ML2 plug-in in the form of mechanism drivers.

Not all Networking plug-ins are compatible with all possible Compute drivers:

Table 7.4. Plug-in compatibility with Compute drivers

Plug-in	Libvirt (KVM/ QEMU)	XenServer	VMware	Hyper-V	Bare-metal
Big Switch / Floodlight	Yes				
Brocade	Yes				
Cisco	Yes				
Cloudbase Hyper-V				Yes	
Linux Bridge	Yes				
Mellanox	Yes				
Midonet	Yes				
ML2	Yes			Yes	
NEC OpenFlow	Yes				
Open vSwitch	Yes				
Plumgrid	Yes		Yes		
Ryu	Yes				
VMware NSX	Yes	Yes	Yes		

Plug-in configurations

For configurations options, see Networking configuration options in *Configuration Reference*. These sections explain how to configure specific plug-ins.

Configure Big Switch, Floodlight REST Proxy plug-in

Procedure 7.2. To use the REST Proxy plug-in with OpenStack Networking

1. Edit the `/etc/neutron/neutron.conf` file and add this line:

```
core_plugin = neutron.plugins.bigswitch.plugin.NeutronRestProxyV2
```

2. Edit the plug-in configuration file, `/etc/neutron/plugins/bigswitch/restproxy.ini`, and specify a comma-separated list of `controller_ip:port` pairs:

```
server = <controller-ip>:<port>
```

For database configuration, see Install Networking Services in the *Installation Guide* in the OpenStack Documentation index. (The link defaults to the Ubuntu version.)

3. Restart `neutron-server` to apply the new settings:

```
# service neutron-server restart
```

Configure Brocade plug-in

Procedure 7.3. To use the Brocade plug-in with OpenStack Networking

1. Install the Brocade-modified Python netconf client (ncclient) library, which is available at https://github.com/brocade/ncclient:

```
$ git clone https://www.github.com/brocade/ncclient
```

As `root` execute:

```
# cd ncclient;python setup.py install
```

2. Edit the `/etc/neutron/neutron.conf` file and set the following option:

```
core_plugin = neutron.plugins.brocade.NeutronPlugin.BrocadePluginV2
```

3. Edit the `/etc/neutron/plugins/brocade/brocade.ini` configuration file for the Brocade plug-in and specify the admin user name, password, and IP address of the Brocade switch:

```
[SWITCH]
username = admin
password = password
address  = switch mgmt ip address
ostype   = NOS
```

For database configuration, see Install Networking Services in any of the *Installation Guides* in the OpenStack Documentation index. (The link defaults to the Ubuntu version.)

4. Restart the `neutron-server` service to apply the new settings:

```
# service neutron-server restart
```

Configure OVS plug-in

If you use the Open vSwitch (OVS) plug-in in a deployment with multiple hosts, you must use either tunneling or vlans to isolate traffic from multiple networks. Tunneling is easier to deploy because it does not require configuring VLANs on network switches.

This procedure uses tunneling:

Procedure 7.4. To configure OpenStack Networking to use the OVS plug-in

1. Edit `/etc/neutron/plugins/openvswitch/ovs_neutron_plugin.ini` to specify these values (for database configuration, see Install Networking Services in *Installation Guide*):

```
enable_tunneling=True
tenant_network_type=gre
tunnel_id_ranges=1:1000
# only required for nodes running agents
local_ip=<data-net-IP-address-of-node>
```

2. If you use the neutron DHCP agent, add these lines to the `/etc/neutron/dhcp_agent.ini` file:

```
dnsmasq_config_file=/etc/neutron/dnsmasq/dnsmasq-neutron.conf
```

3. Create `/etc/neutron/dnsmasq/dnsmasq-neutron.conf`, and add these values to lower the MTU size on instances and prevent packet fragmentation over the GRE tunnel:

```
dhcp-option-force=26,1400
```

4. Restart to apply the new settings:

```
# service neutron-server restart
```

Configure NSX plug-in

Procedure 7.5. To configure OpenStack Networking to use the NSX plug-in

While the instructions in this section refer to the VMware NSX platform, this is formerly known as Nicira NVP.

1. Install the NSX plug-in, as follows:

   ```
   # apt-get install neutron-plugin-vmware
   ```

2. Edit `/etc/neutron/neutron.conf` and set:

   ```
   core_plugin = vmware
   ```

   Example `neutron.conf` file for NSX:

   ```
   core_plugin = vmware
   rabbit_host = 192.168.203.10
   allow_overlapping_ips = True
   ```

3. To configure the NSX controller cluster for the OpenStack Networking Service, locate the `[default]` section in the `/etc/neutron/plugins/vmware/nsx.ini` file, and add the following entries (for database configuration, see Install Networking Services in *Installation Guide*):

   - To establish and configure the connection with the controller cluster you must set some parameters, including NSX API endpoints, access credentials, and settings for HTTP redirects and retries in case of connection failures:

     ```
     nsx_user = <admin user name>
     nsx_password = <password for nsx_user>
     req_timeout = <timeout in seconds for NSX_requests> # default 30 seconds
     http_timeout = <tiemout in seconds for single HTTP request> # default 10
      seconds
     retries = <number of HTTP request retries> # default 2
     redirects = <maximum allowed redirects for a HTTP request> # default 3
     nsx_controllers = <comma separated list of API endpoints>
     ```

     To ensure correct operations, the `nsx_user` user must have administrator credentials on the NSX platform.

     A controller API endpoint consists of the IP address and port for the controller; if you omit the port, port 443 is used. If multiple API endpoints are specified, it is up to the user to ensure that all these endpoints belong to the same controller cluster. The OpenStack Networking VMware NSX plug-in does not perform this check, and results might be unpredictable.

     When you specify multiple API endpoints, the plug-in load-balances requests on the various API endpoints.

   - The UUID of the NSX Transport Zone that should be used by default when a tenant creates a network. You can get this value from the NSX Manager's Transport Zones page:

```
default_tz_uuid = <uuid_of_the_transport_zone>
```

- `default_l3_gw_service_uuid = <uuid_of_the_gateway_service>`

Warning

Ubuntu packaging currently does not update the Neutron init script to point to the NSX configuration file. Instead, you must manually update /etc/default/neutron-server to add this line:

```
NEUTRON_PLUGIN_CONFIG = /etc/neutron/plugins/vmware/nsx.ini
```

4. Restart neutron-server to apply new settings:

```
# service neutron-server restart
```

Example nsx.ini file:

```
[DEFAULT]
default_tz_uuid = d3afb164-b263-4aaa-a3e4-48e0e09bb33c
default_l3_gw_service_uuid=5c8622cc-240a-40a1-9693-e6a5fca4e3cf
nsx_user=admin
nsx_password=changeme
nsx_controllers=10.127.0.100,10.127.0.200:8888
```

Note

To debug nsx.ini configuration issues, run this command from the host that runs neutron-server:

```
# neutron-check-nsx-config <path/to/nsx.ini>
```

This command tests whether neutron-server can log into all of the NSX Controllers and the SQL server, and whether all UUID values are correct.

Load Balancer-as-a-Service and Firewall-as-a-Service

The NSX LBaaS and FWaaS services use the standard OpenStack API with the exception of requiring routed-insertion extension support.

The main differences between the NSX implementation and the community reference implementation of these services are:

1. The NSX LBaaS and FWaaS plug-ins require the routed-insertion extension, which adds the router_id attribute to the VIP (Virtual IP address) and firewall resources and binds these services to a logical router.

2. The community reference implementation of LBaaS only supports a one-arm model, which restricts the VIP to be on the same subnet as the back-end servers. The NSX LBaaS plug-in only supports a two-arm model between north-south traffic, which means that you can create the VIP on only the external (physical) network.

3. The community reference implementation of FWaaS applies firewall rules to all logical routers in a tenant, while the NSX FWaaS plug-in applies firewall rules only to one logical router according to the router_id of the firewall entity.

Procedure 7.6. To configure Load Balancer-as-a-Service and Firewall-as-a-Service with NSX:

1. Edit `/etc/neutron/neutron.conf` file:

   ```
   core_plugin = neutron.plugins.vmware.plugin.NsxServicePlugin
   # Note: comment out service_plug-ins. LBaaS & FWaaS is supported by
    core_plugin NsxServicePlugin
   # service_plugins =
   ```

2. Edit `/etc/neutron/plugins/vmware/nsx.ini` file:

   In addition to the original NSX configuration, the `default_l3_gw_service_uuid` is required for the NSX Advanced plug-in and you must add a `vcns` section:

   ```
   [DEFAULT]
       nsx_password = admin
       nsx_user = admin
       nsx_controllers = 10.37.1.137:443
       default_l3_gw_service_uuid = aae63e9b-2e4e-4efe-81a1-92cf32e308bf
       default_tz_uuid = 2702f27a-869a-49d1-8781-09331a0f6b9e

       [vcns]

       # VSM management URL
       manager_uri = https://10.24.106.219

       # VSM admin user name
       user = admin

       # VSM admin password
       password = default

       # UUID of a logical switch on NSX which has physical network
    connectivity (currently using bridge transport type)
       external_network = f2c023cf-76e2-4625-869b-d0dabcfcc638

       # ID of deployment_container on VSM. Optional, if not specified, a
    default global deployment container is used

       # deployment_container_id =

       # task_status_check_interval configures status check interval for vCNS
    asynchronous API. Default is 2000 msec.

       # task_status_check_interval =
   ```

Configure PLUMgrid plug-in

Procedure 7.7. To use the PLUMgrid plug-in with OpenStack Networking

1. Edit `/etc/neutron/neutron.conf` and set:

   ```
   core_plugin = neutron.plugins.plumgrid.plumgrid_plugin.plumgrid_plugin.
   NeutronPluginPLUMgridV2
   ```

2. Edit `/etc/neutron/plugins/plumgrid/plumgrid.ini` under the `[PLUMgridDirector]` section, and specify the IP address, port, admin user name, and password of the PLUMgrid Director:

```
[PLUMgridDirector]
director_server = "PLUMgrid-director-ip-address"
director_server_port = "PLUMgrid-director-port"
username = "PLUMgrid-director-admin-username"
password = "PLUMgrid-director-admin-password"
```

For database configuration, see Install Networking Services in the *Installation Guide*.

3. Restart `neutron-server` to apply the new settings:

   ```
   # service neutron-server restart
   ```

Configure Ryu plug-in

Procedure 7.8. To use the Ryu plug-in with OpenStack Networking

1. Install the Ryu plug-in, as follows:

   ```
   # apt-get install neutron-plugin-ryu
   ```

2. Edit `/etc/neutron/neutron.conf` and set:

   ```
   core_plugin = neutron.plugins.ryu.ryu_neutron_plugin.RyuNeutronPluginV2
   ```

3. Edit the `/etc/neutron/plugins/ryu/ryu.ini` file and update these options in the `[ovs]` section for the `ryu-neutron-agent`:

   - `openflow_rest_api`. Defines where Ryu is listening for REST API. Substitute `ip-address` and `port-no` based on your Ryu setup.
   - `ovsdb_interface`. Enables Ryu to access the `ovsdb-server`. Substitute `eth0` based on your setup. The IP address is derived from the interface name. If you want to change this value irrespective of the interface name, you can specify `ovsdb_ip`. If you use a non-default port for `ovsdb-server`, you can specify `ovsdb_port`.
   - `tunnel_interface`. Defines which IP address is used for tunneling. If you do not use tunneling, this value is ignored. The IP address is derived from the network interface name.

   For database configuration, see Install Networking Services in *Installation Guide*.

   You can use the same configuration file for many compute nodes by using a network interface name with a different IP address:

   ```
   openflow_rest_api = <ip-address>:<port-no> ovsdb_interface = <eth0>
    tunnel_interface = <eth0>
   ```

4. Restart `neutron-server` to apply the new settings:

   ```
   # service neutron-server restart
   ```

Configure neutron agents

Plug-ins typically have requirements for particular software that must be run on each node that handles data packets. This includes any node that runs `nova-compute` and nodes that run dedicated OpenStack Networking service agents such as `neutron-dhcp-agent`, `neutron-l3-agent`, `neutron-metering-agent` or `neutron-lbaas-agent`.

A data-forwarding node typically has a network interface with an IP address on the “management network” and another interface on the “data network”.

This section shows you how to install and configure a subset of the available plug-ins, which might include the installation of switching software (for example, Open vSwitch) and as agents used to communicate with the `neutron-server` process running elsewhere in the data center.

Configure data-forwarding nodes

Node set up: OVS plug-in

Note

This section also applies to the ML2 plug-in when Open vSwitch is used as a mechanism driver.

If you use the Open vSwitch plug-in, you must install Open vSwitch and the `neutron-plugin-openvswitch-agent` agent on each data-forwarding node:

Warning

Do not install the openvswitch-brcompat package because it prevents the security group functionality from operating correctly.

Procedure 7.9. To set up each node for the OVS plug-in

1. Install the OVS agent package. This action also installs the Open vSwitch software as a dependency:

   ```
   # apt-get install neutron-plugin-openvswitch-agent
   ```

2. On each node that runs the `neutron-plugin-openvswitch-agent`, complete these steps:

   - Replicate the `ovs_neutron_plugin.ini` file that you created on the node.
   - If you use tunneling, update the `ovs_neutron_plugin.ini` file for the node with the IP address that is configured on the data network for the node by using the `local_ip` value.

3. Restart Open vSwitch to properly load the kernel module:

   ```
   # service openvswitch-switch restart
   ```

4. Restart the agent:

   ```
   # service neutron-plugin-openvswitch-agent restart
   ```

5. All nodes that run `neutron-plugin-openvswitch-agent` must have an OVS `br-int` bridge. To create the bridge, run:

   ```
   # ovs-vsctl add-br br-int
   ```

Node set up: NSX plug-in

If you use the NSX plug-in, you must also install Open vSwitch on each data-forwarding node. However, you do not need to install an additional agent on each node.

Warning

It is critical that you are running an Open vSwitch version that is compatible with the current version of the NSX Controller software. Do not use the Open vSwitch version that is installed by default on Ubuntu. Instead, use the Open vSwitch version that is provided on the VMware support portal for your NSX Controller version.

Procedure 7.10. To set up each node for the NSX plug-in

1. Ensure that each data-forwarding node has an IP address on the management network, and an IP address on the "data network" that is used for tunneling data traffic. For full details on configuring your forwarding node, see the *NSX Administrator Guide*.

2. Use the *NSX Administrator Guide* to add the node as a Hypervisor by using the NSX Manager GUI. Even if your forwarding node has no VMs and is only used for services agents like `neutron-dhcp-agent` or `neutron-lbaas-agent`, it should still be added to NSX as a Hypervisor.

3. After following the *NSX Administrator Guide*, use the page for this Hypervisor in the NSX Manager GUI to confirm that the node is properly connected to the NSX Controller Cluster and that the NSX Controller Cluster can see the `br-int` integration bridge.

Node set up: Ryu plug-in

If you use the Ryu plug-in, you must install both Open vSwitch and Ryu, in addition to the Ryu agent package:

Procedure 7.11. To set up each node for the Ryu plug-in

1. Install Ryu (there isn't currently an Ryu package for Ubuntu):

```
# pip install ryu
```

2. Install the Ryu agent and Open vSwitch packages:

```
# apt-get install neutron-plugin-ryu-agent openvswitch-switch python-
openvswitch openvswitch-datapath-dkms
```

3. Replicate the `ovs_ryu_plugin.ini` and `neutron.conf` files created in the above step on all nodes running `neutron-plugin-ryu-agent`.

4. Restart Open vSwitch to properly load the kernel module:

```
# service openvswitch-switch restart
```

5. Restart the agent:

```
# service neutron-plugin-ryu-agent restart
```

6. All nodes running `neutron-plugin-ryu-agent` also require that an OVS bridge named "br-int" exists on each node. To create the bridge, run:

```
# ovs-vsctl add-br br-int
```

Configure DHCP agent

The DHCP service agent is compatible with all existing plug-ins and is required for all deployments where VMs should automatically receive IP addresses through DHCP.

Procedure 7.12. To install and configure the DHCP agent

1. You must configure the host running the `neutron-dhcp-agent` as a "data forwarding node" according to the requirements for your plug-in (see the section called "Configure neutron agents" [178]).

2. Install the DHCP agent:

```
# apt-get install neutron-dhcp-agent
```

3. Finally, update any options in the `/etc/neutron/dhcp_agent.ini` file that depend on the plug-in in use (see the sub-sections).

Important

If you reboot a node that runs the DHCP agent, you must run the **neutron-ovs-cleanup** command before the `neutron-dhcp-agent` service starts.

On Red Hat, SUSE, and Ubuntu based systems, the `neutron-ovs-cleanup` service runs the **neutron-ovs-cleanup** command automatically. However, on Debian-based systems (including Ubuntu prior to Icehouse), you must manually run this command or write your own system script that runs on boot before the `neutron-dhcp-agent` service starts.

DHCP agent setup: OVS plug-in

These DHCP agent options are required in the `/etc/neutron/dhcp_agent.ini` file for the OVS plug-in:

```
[DEFAULT]
enable_isolated_metadata = True
use_namespaces = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

DHCP agent setup: NSX plug-in

These DHCP agent options are required in the `/etc/neutron/dhcp_agent.ini` file for the NSX plug-in:

```
[DEFAULT]
enable_metadata_network = True
enable_isolated_metadata = True
use_namespaces = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

DHCP agent setup: Ryu plug-in

These DHCP agent options are required in the `/etc/neutron/dhcp_agent.ini` file for the Ryu plug-in:

```
[DEFAULT]
use_namespace = True
interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
```

Configure L3 agent

The OpenStack Networking Service has a widely used API extension to allow administrators and tenants to create routers to interconnect L2 networks, and floating IPs to make ports on private networks publicly accessible.

Many plug-ins rely on the L3 service agent to implement the L3 functionality. However, the following plug-ins already have built-in L3 capabilities:

- NSX plug-in
- Big Switch/Floodlight plug-in, which supports both the open source Floodlight controller and the proprietary Big Switch controller.

  Note

  Only the proprietary BigSwitch controller implements L3 functionality. When using Floodlight as your OpenFlow controller, L3 functionality is not available.

- PLUMgrid plug-in

Warning

Do not configure or use `neutron-l3-agent` if you use one of these plug-ins.

Procedure 7.13. To install the L3 agent for all other plug-ins

1. Install the `neutron-l3-agent` binary on the network node:

   ```
   # apt-get install neutron-l3-agent
   ```

2. To uplink the node that runs `neutron-l3-agent` to the external network, create a bridge named "br-ex" and attach the NIC for the external network to this bridge.

   For example, with Open vSwitch and NIC eth1 connected to the external network, run:

   ```
   # ovs-vsctl add-br br-ex
   # ovs-vsctl add-port br-ex eth1
   ```

   Do not manually configure an IP address on the NIC connected to the external network for the node running `neutron-l3-agent`. Rather, you must have a range of IP addresses from the external network that can be used by OpenStack Networking for routers that uplink to the external network. This range must be large enough to have an IP address for each router in the deployment, as well as each floating IP.

3. The `neutron-l3-agent` uses the Linux IP stack and iptables to perform L3 forwarding and NAT. In order to support multiple routers with potentially overlapping IP addresses, `neutron-l3-agent` defaults to using Linux network namespaces to provide isolated forwarding contexts. As a result, the IP addresses of routers are not visible simply by running the **ip addr list** or **ifconfig** command on the node. Similarly, you cannot directly **ping** fixed IPs.

   To do either of these things, you must run the command within a particular network namespace for the router. The namespace has the name "qrouter-<UUID of the router>. These example commands run in the router namespace with UUID 47af3868-0fa8-4447-85f6-1304de32153b:

   ```
   # ip netns exec qrouter-47af3868-0fa8-4447-85f6-1304de32153b ip addr list
   ```

   ```
   # ip netns exec qrouter-47af3868-0fa8-4447-85f6-1304de32153b ping <fixed-
   ip>
   ```

Important

If you reboot a node that runs the L3 agent, you must run the **neutron-ovs-cleanup** command before the `neutron-l3-agent` service starts.

On Red Hat, SUSE and Ubuntu based systems, the `neutron-ovs-cleanup` service runs the **neutron-ovs-cleanup** command automatically. However, on Debian-based systems (including Ubuntu prior to Icehouse), you must manually run this command or write your own system script that runs on boot before the `neutron-l3-agent` service starts.

Configure metering agent

Starting with the Havana release, the Neutron Metering resides beside `neutron-l3-agent`.

Procedure 7.14. To install the metering agent and configure the node

1. Install the agent by running:

   ```
   # apt-get install neutron-metering-agent
   ```

Package name prior to Icehouse

In releases of Neutron prior to Icehouse, this package was named neutron-plugin-metering-agent.

2. If you use one of the following plugins, you need to configure the metering agent with these lines as well:

   - An OVS-based plug-in such as OVS, NSX, Ryu, NEC, BigSwitch/Floodlight:

     ```
     interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
     ```

   - A plug-in that uses LinuxBridge:

     ```
     interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
     ```

3. To use the reference implementation, you must set:

   ```
   driver = neutron.services.metering.drivers.iptables.iptables_driver.
   IptablesMeteringDriver
   ```

4. Set this parameter in the `neutron.conf` file on the host that runs `neutron-server`:

   ```
   service_plugins = neutron.services.metering.metering_plugin.MeteringPlugin
   ```

Configure Load-Balancing-as-a-Service (LBaaS)

Configure Load-Balancing-as-a-Service (LBaas) with the Open vSwitch or Linux Bridge plug-in. The Open vSwitch LBaaS driver is required when enabling LBaaS for OVS-based plug-ins, including BigSwitch, Floodlight, NEC, NSX, and Ryu.

1. Install the agent:

   ```
   # apt-get install neutron-lbaas-agent
   ```

2. Enable the HAProxy plug-in using the `service_provider` parameter in the `/etc/neutron/neutron.conf` file:

   ```
   service_provider = LOADBALANCER:Haproxy:neutron.services.loadbalancer.
   drivers.haproxy.plugin_driver.HaproxyOnHostPluginDriver:default
   ```

3. Enable the load balancer plugin using `service_plugin` in the `/etc/neutron/neutron.conf` file:

   ```
   service_plugins = neutron.services.loadbalancer.plugin.LoadBalancerPlugin
   ```

4. Enable the HAProxy load balancer in the `/etc/neutron/lbaas_agent.ini` file:

   ```
   device_driver = neutron.services.loadbalancer.drivers.haproxy.
   namespace_driver.HaproxyNSDriver
   ```

5. Select the required driver in the `/etc/neutron/lbaas_agent.ini` file:

   Enable the Open vSwitch LBaaS driver:

   ```
   interface_driver = neutron.agent.linux.interface.OVSInterfaceDriver
   ```

   Or enable the Linux Bridge LBaaS driver:

   ```
   interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
   ```

   Apply the new settings by restarting the `neutron-server` and `neutron-lbaas-agent` services.

   ### Upgrade from Havana to Icehouse

   There were changes in LBaaS server-agent communications in Icehouse so during Havana to Icehouse transition make sure to upgrade both server and agent sides before actual use of the load balancing service.

6. Enable Load Balancing in the **Project** section of the Dashboard user interface:

Change the `enable_lb` option to *True* in the `/etc/openstack-dashboard/local_settings` file:

```
OPENSTACK_NEUTRON_NETWORK = {'enable_lb': True,
```

Apply the new settings by restarting the `httpd` service. You can now view the Load Balancer management options in dashboard's **Project** view.

Networking architecture

Before you deploy Networking, it's useful to understand the Networking services and how they interact with the OpenStack components.

Overview

Networking is a standalone component in the OpenStack modular architecture. It's positioned alongside OpenStack components such as Compute, Image Service, Identity, or the Dashboard. Like those components, a deployment of Networking often involves deploying several services to a variety of hosts.

The Networking server uses the `neutron-server` daemon to expose the Networking API and enable administration of the configured Networking plug-in. Typically, the plug-in requires access to a database for persistent storage (also similar to other OpenStack services).

If your deployment uses a controller host to run centralized Compute components, you can deploy the Networking server to that same host. However, Networking is entirely standalone and can be deployed to a dedicated host. Depending on your configuration, Networking can also include the following agents:

Table 7.5. Networking agents

Agent	Description
plug-in agent (`neutron-*-agent`)	Runs on each hypervisor to perform local vSwitch configuration. The agent that runs, depends on the plug-in that you use. Certain plug-ins do not require an agent.
dhcp agent (`neutron-dhcp-agent`)	Provides DHCP services to tenant networks. Required by certain plug-ins.
l3 agent (`neutron-l3-agent`)	Provides L3/NAT forwarding to provide external network access for VMs on tenant networks. Required by certain plug-ins.
metering agent (`neutron-metering-agent`)	Provides L3 traffic metering for tenant networks.

These agents interact with the main neutron process through RPC (for example, RabbitMQ or Qpid) or through the standard Networking API. In addition, Networking integrates with OpenStack components in a number of ways:

- Networking relies on the Identity service (Keystone) for the authentication and authorization of all API requests.
- Compute (Nova) interacts with Networking through calls to its standard API. As part of creating a VM, the `nova-compute` service communicates with the Networking API to plug each virtual NIC on the VM into a particular network.

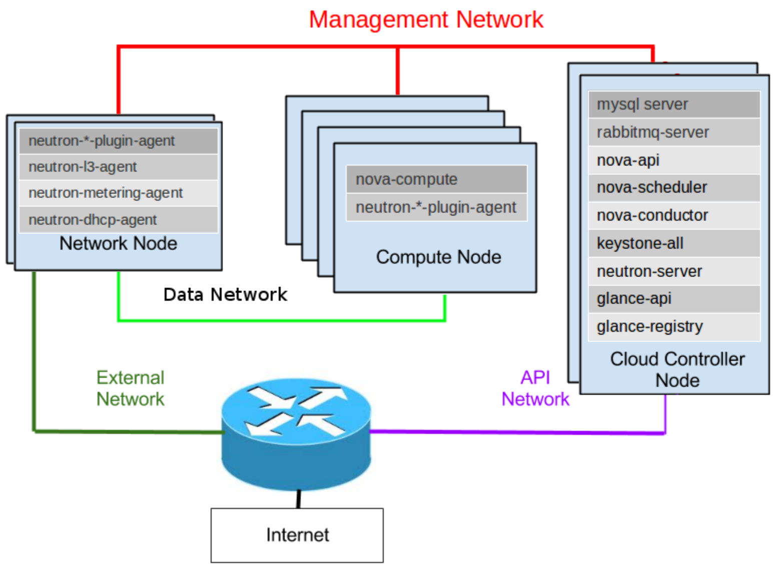
- The Dashboard (Horizon) integrates with the Networking API, enabling administrators and tenant users to create and manage network services through a web-based GUI.

Place services on physical hosts

Like other OpenStack services, Networking enables you to run services across multiple physical devices. It is also possible to run all service daemons on a single physical host for evaluation purposes. Alternatively, you can run each service on a dedicated physical host and replicate certain services across multiple hosts for redundancy purposes. For more information, see the *OpenStack Configuration Reference*.

A standard architectural design includes a cloud controller host, a network gateway host, and a number of hypervisors for hosting virtual machines. The cloud controller and network gateway can be on the same host. However, if you expect VMs to send significant traffic to or from the Internet, a dedicated network gateway host helps avoid CPU contention between the `neutron-l3-agent` and other OpenStack services that forward packets.

Network connectivity for physical hosts



A standard Networking deployment includes one or more of the following physical networks:

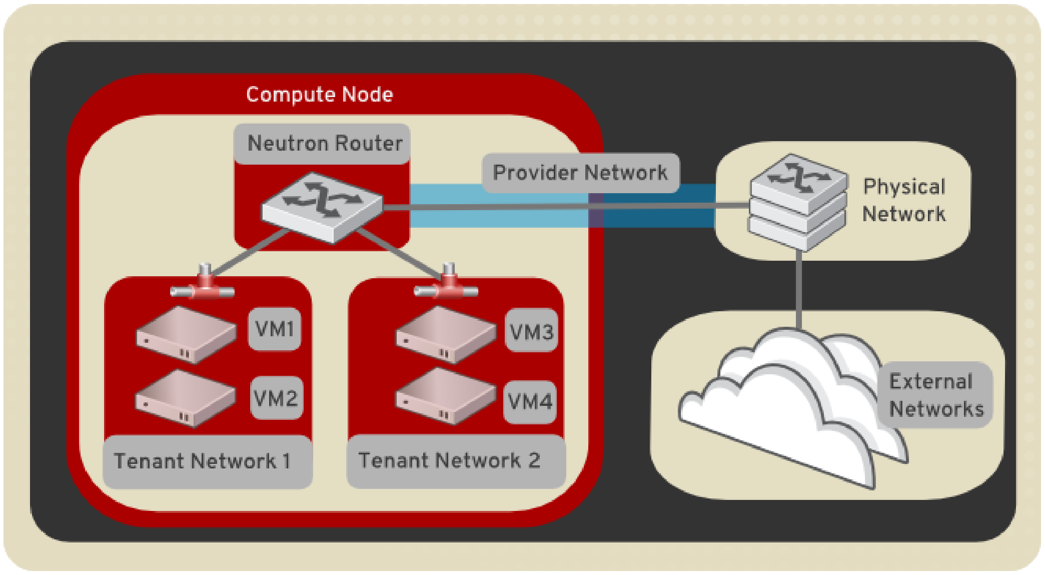
Table 7.6. General distinct physical data center networks

Network	Description
Management network	Provides internal communication between OpenStack components. IP addresses on this network should be reachable only within the data center.
Data network	Provides VM data communication within the cloud deployment. The IP addressing requirements of this network depend on the Networking plug-in that is used.
External network	Provides VMs with Internet access in some deployment scenarios. Anyone on the Internet can reach IP addresses on this network.
API network	Exposes all OpenStack APIs, including the Networking API, to tenants. IP addresses on this network should be reachable by anyone on the Internet. The API network might be the same as the external network, because it is possible to create an external-network subnet that has allocated IP ranges that use less than the full range of IP addresses in an IP block.

Tenant and provider networks

The following diagram presents an overview of the tenant and provider network types, and illustrates how they interact within the overall Networking topology:

Figure 7.2. Tenant and provider networks



Tenant networks. Tenant networks are created by users for connectivity within projects; they are fully isolated by default and are not shared with other projects. Networking supports a range of tenant network types:

Flat	All instances reside on the same network, which can also be shared with the hosts. No VLAN tagging or other network segregation takes place.
Local	Instances reside on the local compute host and are effectively isolated from any external networks.
VLAN	Networking allows users to create multiple provider or tenant networks using VLAN IDs (802.1Q tagged) that correspond to VLANs present in the physical network. This allows instances to communicate with each other across the environment. They can also communicate with dedicated servers, firewalls, load balancers and other networking infrastructure on the same layer 2 VLAN.
VXLAN and GRE	VXLAN and GRE use network overlays to support private communication between instances. A Networking router is required to enable traffic to traverse outside of the GRE or VXLAN tenant network. A router is also required to connect directly-connected tenant networks with external networks, including the Internet; the router provides the ability to connect to instances directly from an external network using floating IP addresses.

Provider networks. Provider networks are created by the OpenStack administrator and map directly to an existing physical network in the data center. Useful network types in this category are flat (untagged) and VLAN (802.1Q tagged). It is possible to allow provider networks to be shared among tenants as part of the network creation process.

Configure Identity Service for Networking

Procedure 7.15. To configure the Identity Service for use with Networking

1. **Create the `get_id()` function**

   The `get_id()` function stores the ID of created objects, and removes the need to copy and paste object IDs in later steps:

   a. Add the following function to your `.bashrc` file:

   ```
   function get_id () {
   echo `"$@" | awk '/ id / { print $4 }'`
   }
   ```

   b. Source the `.bashrc` file:

   ```
   $ source .bashrc
   ```

2. **Create the Networking service entry**

   Networking must be available in the Compute service catalog. Create the service:

   ```
   $ NEUTRON_SERVICE_ID=$(get_id keystone service-create --name neutron --
   type network --description 'OpenStack Networking Service')
   ```

3. **Create the Networking service endpoint entry**

   The way that you create a Networking endpoint entry depends on whether you are using the SQL or the template catalog driver:

   - If you use the *SQL driver*, run the following command with the specified region (`$REGION`), IP address of the Networking server (`$IP`), and service ID (`$NEUTRON_SERVICE_ID`, obtained in the previous step).

     ```
     $ keystone endpoint-create --region $REGION --service-id
      $NEUTRON_SERVICE_ID \
        --publicurl 'http://$IP:9696/' --adminurl 'http://$IP:9696/' --
     internalurl 'http://$IP:9696/'
     ```

     For example:

     ```
     $ keystone endpoint-create --region myregion --service-id
      $NEUTRON_SERVICE_ID \
        --publicurl "http://10.211.55.17:9696/" --adminurl "http://10.211.55.
     17:9696/" --internalurl "http://10.211.55.17:9696/"
     ```

   - If you are using the *template driver*, specify the following parameters in your Compute catalog template file (`default_catalog.templates`), along with the region (`$REGION`) and IP address of the Networking server (`$IP`).

```
catalog.$REGION.network.publicURL = http://$IP:9696
catalog.$REGION.network.adminURL = http://$IP:9696
catalog.$REGION.network.internalURL = http://$IP:9696
catalog.$REGION.network.name = Network Service
```

For example:

```
catalog.$Region.network.publicURL = http://10.211.55.17:9696
catalog.$Region.network.adminURL = http://10.211.55.17:9696
catalog.$Region.network.internalURL = http://10.211.55.17:9696
catalog.$Region.network.name = Network Service
```

4. **Create the Networking service user**

   You must provide admin user credentials that Compute and some internal Networking components can use to access the Networking API. Create a special `service` tenant and a `neutron` user within this tenant, and assign an `admin` role to this role.

   a. Create the `admin` role:

   ```
   $ ADMIN_ROLE=$(get_id keystone role-create --name=admin)
   ```

   b. Create the `neutron` user:

   ```
   $ NEUTRON_USER=$(get_id keystone user-create --name=neutron --pass=
   "$NEUTRON_PASSWORD" --email=demo@example.com --tenant-id service)
   ```

   c. Create the `service` tenant:

   ```
   $ SERVICE_TENANT=$(get_id keystone tenant-create --name service --
   description "Services Tenant")
   ```

   d. Establish the relationship among the tenant, user, and role:

   ```
   $ keystone user-role-add --user_id $NEUTRON_USER --role_id $ADMIN_ROLE
    --tenant_id $SERVICE_TENANT
   ```

For information about how to create service entries and users, see the *OpenStack Installation Guide* for your distribution (docs.openstack.org).

Compute

If you use Networking, do not run the Compute `nova-network` service (like you do in traditional Compute deployments). Instead, Compute delegates most network-related decisions to Networking. Compute proxies tenant-facing API calls to manage security groups and floating IPs to Networking APIs. However, operator-facing tools such as `nova-manage`, are not proxied and should not be used.

Warning

When you configure networking, you must use this guide. Do not rely on Compute networking documentation or past experience with Compute. If a **nova** command or configuration option related to networking is not mentioned in this guide, the command is probably not supported for use with Networking. In particular, you cannot use CLI tools like **nova-manage** and **nova** to manage networks or IP addressing, including both fixed and floating IPs, with Networking.

Note

Uninstall `nova-network` and reboot any physical nodes that have been running `nova-network` before using them to run Networking. Inadvertently running the `nova-network` process while using Networking can cause problems, as can stale iptables rules pushed down by previously running `nova-network`.

To ensure that Compute works properly with Networking (rather than the legacy `nova-network` mechanism), you must adjust settings in the `nova.conf` configuration file.

Networking API and credential configuration

Each time you provision or de-provision a VM in Compute, `nova-*` services communicate with Networking using the standard API. For this to happen, you must configure the following items in the `nova.conf` file (used by each `nova-compute` and `nova-api` instance).

Table 7.7. nova.conf API and credential settings

Item	Configuration
`network_api_class`	Modify from the default to `nova.network.neutronv2.api.API`, to indicate that Networking should be used rather than the traditional `nova-network` networking model.
`neutron_url`	Update to the hostname/IP and port of the `neutron-server` instance for this deployment.
`neutron_auth_strategy`	Keep the default `keystone` value for all production deployments.
`neutron_admin_tenant_name`	Update to the name of the service tenant created in the above section on Identity configuration.
`neutron_admin_username`	Update to the name of the user created in the above section on Identity configuration.
`neutron_admin_password`	Update to the password of the user created in the above section on Identity configuration.
`neutron_admin_auth_url`	Update to the Identity server IP and port. This is the Identity (keystone) admin API server IP and port value, and not the Identity service API IP and port.

Configure security groups

The Networking Service provides security group functionality using a mechanism that is more flexible and powerful than the security group capabilities built into Compute. Therefore, if you use Networking, you should always disable built-in security groups and proxy all security group calls to the Networking API . If you do not, security policies will conflict by being simultaneously applied by both services.

To proxy security groups to Networking, use the following configuration values in `nova.conf`:

Table 7.8. nova.conf security group settings

Item	Configuration
`firewall_driver`	Update to `nova.virt.firewall.NoopFirewallDriver`, so that `nova-compute` does not perform iptables-based filtering itself.
`security_group_api`	Update to `neutron`, so that all security group requests are proxied to the Network Service.

Configure metadata

The Compute service allows VMs to query metadata associated with a VM by making a web request to a special 169.254.169.254 address. Networking supports proxying those requests to `nova-api`, even when the requests are made from isolated networks, or from multiple networks that use overlapping IP addresses.

To enable proxying the requests, you must update the following fields in `nova.conf`.

Table 7.9. nova.conf metadata settings

Item	Configuration
`service_neutron_metadata_proxy`	Update to `true` otherwise `nova-api` will not properly respond to requests from the `neutron-metadata-agent`.
`neutron_metadata_proxy_shared_secret`	Update the string "password" value. You must also configure the same value in the `metadata_agent.ini` file, to authenticate requests made for metadata. The default value of an empty string in both files will allow metadata to function, but will not be secure if any non-trusted entities have access to the metadata APIs exposed by `nova-api`.

Note

As a precaution, even when using `neutron_metadata_proxy_shared_secret`, it is recommended that you do not expose metadata using the same `nova-api` instances that are used for tenants. Instead, you should run a dedicated set of `nova-api` instances for metadata that are available only on your management network. Whether a given `nova-api` instance exposes metadata APIs is determined by the value of `enabled_apis` in its `nova.conf`.

Example nova.conf (for `nova-compute` and `nova-api`)

Example values for the above settings, assuming a cloud controller node running Compute and Networking with an IP address of 192.168.1.2:

```
network_api_class=nova.network.neutronv2.api.API
neutron_url=http://192.168.1.2:9696
neutron_auth_strategy=keystone
neutron_admin_tenant_name=service
neutron_admin_username=neutron
neutron_admin_password=password
neutron_admin_auth_url=http://192.168.1.2:35357/v2.0

security_group_api=neutron
firewall_driver=nova.virt.firewall.NoopFirewallDriver

service_neutron_metadata_proxy=true
neutron_metadata_proxy_shared_secret=foo
```

Networking scenarios

This chapter describes two networking scenarios and how the Open vSwitch plug-in and the Linux Bridge plug-in implement these scenarios.

Open vSwitch

This section describes how the Open vSwitch plug-in implements the Networking abstractions.

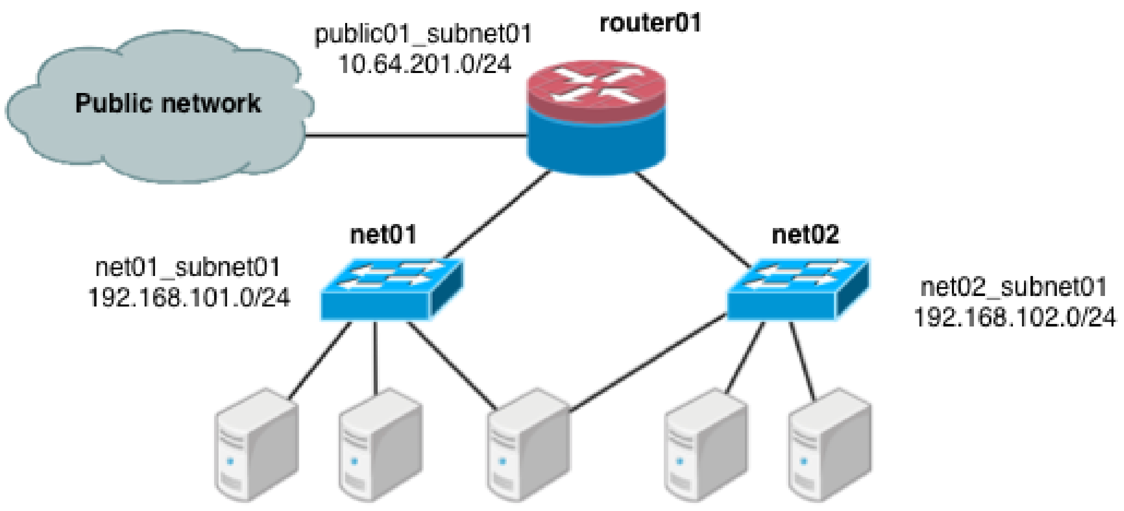
Configuration

This example uses VLAN segmentation on the switches to isolate tenant networks. This configuration labels the physical network associated with the public network as `physnet1`, and the physical network associated with the data network as `physnet2`, which leads to the following configuration options in `ovs_neutron_plugin.ini`:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet2:100:110
integration_bridge = br-int
bridge_mappings = physnet2:br-eth1
```

Scenario 1: one tenant, two networks, one router

The first scenario has two private networks (`net01`, and `net02`), each with one subnet (`net01_subnet01`: 192.168.101.0/24, `net02_subnet01`, 192.168.102.0/24). Both private networks are attached to a router that connects them to the public network (10.64.201.0/24).



Under the `service` tenant, create the shared router, define the public network, and set it as the default gateway of the router

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ neutron router-create router01
$ neutron net-create --tenant-id $tenant public01 \
          --provider:network_type flat \
          --provider:physical_network physnet1 \
          --router:external=True
$ neutron subnet-create --tenant-id $tenant --name public01_subnet01 \
          --gateway 10.64.201.254 public01 10.64.201.0/24 --disable-dhcp
$ neutron router-gateway-set router01 public01
```

Under the demo user tenant, create the private network net01 and corresponding subnet, and connect it to the router01 router. Configure it to use VLAN ID 101 on the physical switch.

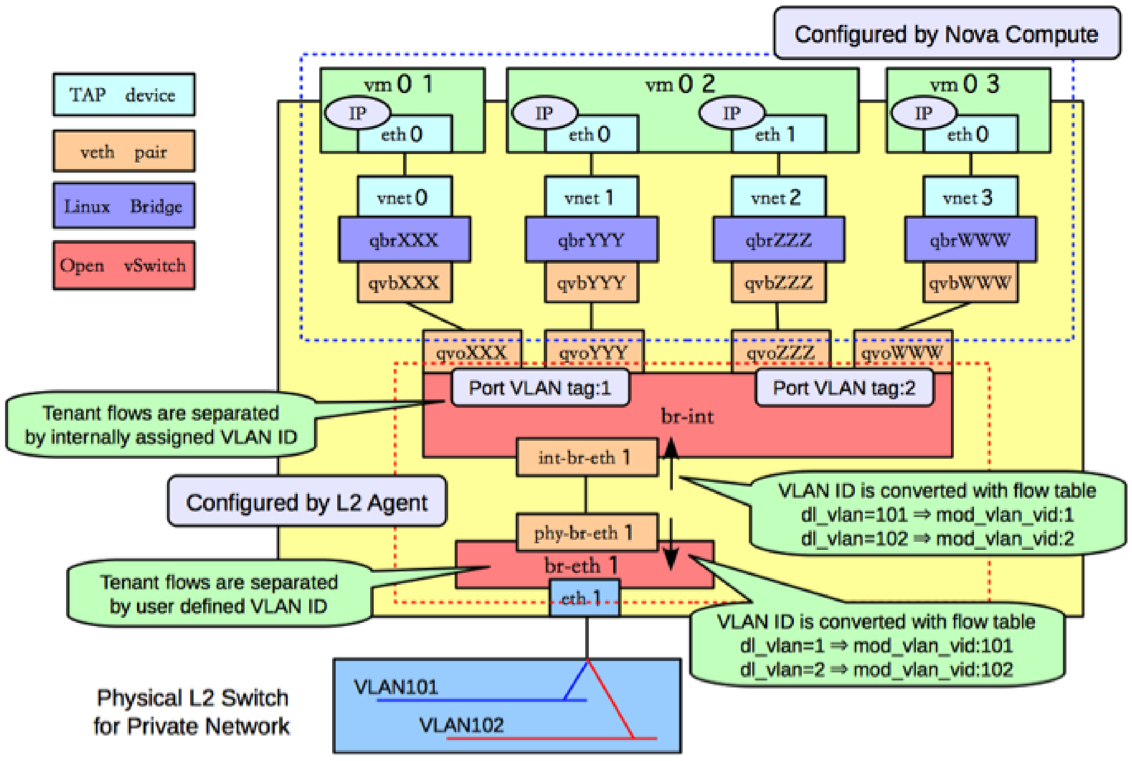
```
$ tenant=$(keystone tenant-list|awk '/demo/ {print $2}'
$ neutron net-create --tenant-id $tenant net01 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 101
$ neutron subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.
168.101.0/24
$ neutron router-interface-add router01 net01_subnet01
```

Similarly, for net02, using VLAN ID 102 on the physical switch:

```
$ neutron net-create --tenant-id $tenant net02 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 102
$ neutron subnet-create --tenant-id $tenant --name net02_subnet01 net02 192.
168.102.0/24
$ neutron router-interface-add router01 net02_subnet01
```

Scenario 1: Compute host config

The following figure shows how to configure various Linux networking devices on the compute host:


Configured by Nova Compute
TAP device
veth pair
Linux Bridge
Open vSwitch
vm01
vm02
vm03
IP
eth0
eth1
vnet0
vnet1
vnet2
vnet3
qbrXXX
qbrYYY
qbrZZZ
qbrWWW
qvbXXX
qvbYYY
qvbZZZ
qvbWWW
qvoXXX
qvoYYY
qvoZZZ
qvoWWW
Port VLAN tag:1
Port VLAN tag:2
br-int
Tenant flows are separated by internally assigned VLAN ID
int-br-eth1
Configured by L2 Agent
VLAN ID is converted with flow table dl_vlan=101 ⇒ mod_vlan_vid:1 dl_vlan=102 ⇒ mod_vlan_vid:2
phy-br-eth1
br-eth1
eth1
Tenant flows are separated by user defined VLAN ID
VLAN ID is converted with flow table dl_vlan=1 ⇒ mod_vlan_vid:101 dl_vlan=2 ⇒ mod_vlan_vid:102
Physical L2 Switch for Private Network
VLAN101
VLAN102

Types of network devices

Note

There are four distinct type of virtual networking devices: TAP devices, veth pairs, Linux bridges, and Open vSwitch bridges. For an ethernet frame to travel from `eth0` of virtual machine `vm01` to the physical network, it must pass through nine devices inside of the host: TAP `vnet0`, Linux bridge `qbrnnn`, veth pair `(qvbnnn, qvonnn)`, Open vSwitch bridge `br-int`, veth pair `(int-br-eth1, phy-br-eth1)`, and, finally, the physical network interface card `eth1`.

A *TAP device*, such as `vnet0` is how hypervisors such as KVM and Xen implement a virtual network interface card (typically called a VIF or vNIC). An ethernet frame sent to a TAP device is received by the guest operating system.

A *veth pair* is a pair of directly connected virtual network interfaces. An ethernet frame sent to one end of a veth pair is received by the other end of a veth pair. Networking uses veth pairs as virtual patch cables to make connections between virtual bridges.

A *Linux bridge* behaves like a hub: you can connect multiple (physical or virtual) network interfaces devices to a Linux bridge. Any ethernet frames that come in from one interface attached to the bridge is transmitted to all of the other devices.

An *Open vSwitch bridge* behaves like a virtual switch: network interface devices connect to Open vSwitch bridge's ports, and the ports can be configured much like a physical switch's ports, including VLAN configurations.

Integration bridge

The `br-int` Open vSwitch bridge is the integration bridge: all guests running on the compute host connect to this bridge. Networking implements isolation across these guests by configuring the `br-int` ports.

Physical connectivity bridge

The `br-eth1` bridge provides connectivity to the physical network interface card, `eth1`. It connects to the integration bridge by a veth pair: `(int-br-eth1, phy-br-eth1)`.

VLAN translation

In this example, net01 and net02 have VLAN ids of 1 and 2, respectively. However, the physical network in our example only supports VLAN IDs in the range 101 through 110. The Open vSwitch agent is responsible for configuring flow rules on `br-int` and `br-eth1` to do VLAN translation. When `br-eth1` receives a frame marked with VLAN ID 1 on the port associated with `phy-br-eth1`, it modifies the VLAN ID in the frame to 101. Similarly, when `br-int` receives a frame marked with VLAN ID 101 on the port associated with `int-br-eth1`, it modifies the VLAN ID in the frame to 1.

Security groups: iptables and Linux bridges

Ideally, the TAP device `vnet0` would be connected directly to the integration bridge, `br-int`. Unfortunately, this isn't possible because of how OpenStack security groups are currently implemented. OpenStack uses iptables rules on the TAP devices such as `vnet0` to implement security groups, and Open vSwitch is not compatible with iptables rules that are applied directly on TAP devices that are connected to an Open vSwitch port.

Networking uses an extra Linux bridge and a veth pair as a workaround for this issue. Instead of connecting `vnet0` to an Open vSwitch bridge, it is connected to a Linux bridge, `qbr`*`XXX`*. This bridge is connected to the integration bridge, `br-int`, through the `(qvb`*`XXX`*`, qvo`*`XXX`*`)` veth pair.

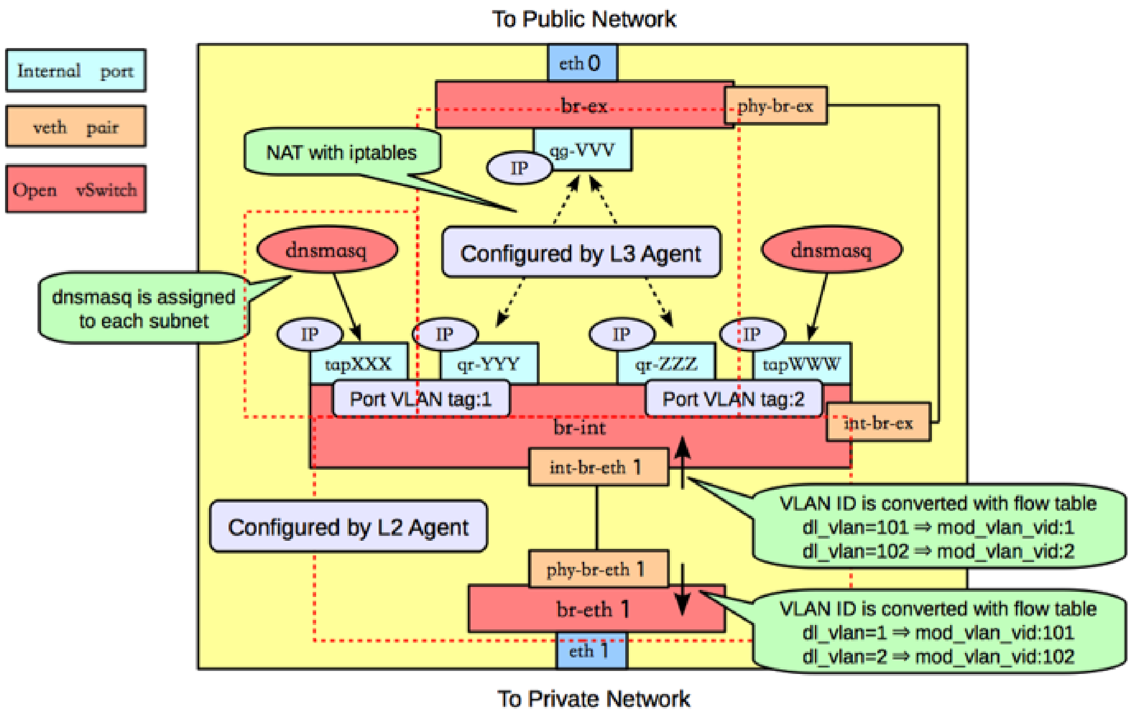
Scenario 1: Network host config

The network host runs the neutron-openvswitch-plugin-agent, the neutron-dhcp-agent, neutron-l3-agent, and neutron-metadata-agent services.

On the network host, assume that eth0 is connected to the external network, and eth1 is connected to the data network, which leads to the following configuration in the `ovs_neutron_plugin.ini` file:

```
[ovs]
tenant_network_type = vlan
network_vlan_ranges = physnet2:101:110
integration_bridge = br-int
bridge_mappings = physnet1:br-ex,physnet2:br-eth1
```

The following figure shows the network devices on the network host:



As on the compute host, there is an Open vSwitch integration bridge (`br-int`) and an Open vSwitch bridge connected to the data network (`br-eth1`), and the two are connected by a veth pair, and the neutron-openvswitch-plugin-agent configures the ports on both switches to do VLAN translation.

An additional Open vSwitch bridge, `br-ex`, connects to the physical interface that is connected to the external network. In this example, that physical interface is `eth0`.

Note

While the integration bridge and the external bridge are connected by a veth pair `(int-br-ex, phy-br-ex)`, this example uses layer 3 connectivity to route packets from the internal networks to the public network: no packets traverse that veth pair in this example.

Open vSwitch internal ports

The network host uses Open vSwitch *internal ports*. Internal ports enable you to assign one or more IP addresses to an Open vSwitch bridge. In previous example, the `br-int` bridge has four internal ports: `tap`*XXX*, `qr-`*YYY*, `qr-`*ZZZ*, and `tap`*WWW*. Each internal port has a separate IP address associated with it. An internal port, `qg-`VVV, is on the `br-ex` bridge.

DHCP agent

By default, The Networking DHCP agent uses a process called dnsmasq to provide DHCP services to guests. Networking must create an internal port for each network that requires DHCP services and attach a dnsmasq process to that port. In the previous example, the `tap`*XXX* interface is on `net01_subnet01`, and the `tap`*WWW* interface is on `net02_subnet01`.

L3 agent (routing)

The Networking L3 agent uses Open vSwitch internal ports to implement routing and relies on the network host to route the packets across the interfaces. In this example, the `qr-`YYY interface is on `net01_subnet01` and has the IP address 192.168.101.1/24. The `qr-`*ZZZ*, interface is on `net02_subnet01` and has the IP address `192.168.102.1/24`. The `qg-`*VVV* interface has the IP address `10.64.201.254/24`. Because each of these interfaces is visible to the network host operating system, the network host routes the packets across the interfaces, as long as an administrator has enabled IP forwarding.

The L3 agent uses iptables to implement floating IPs to do the network address translation (NAT).

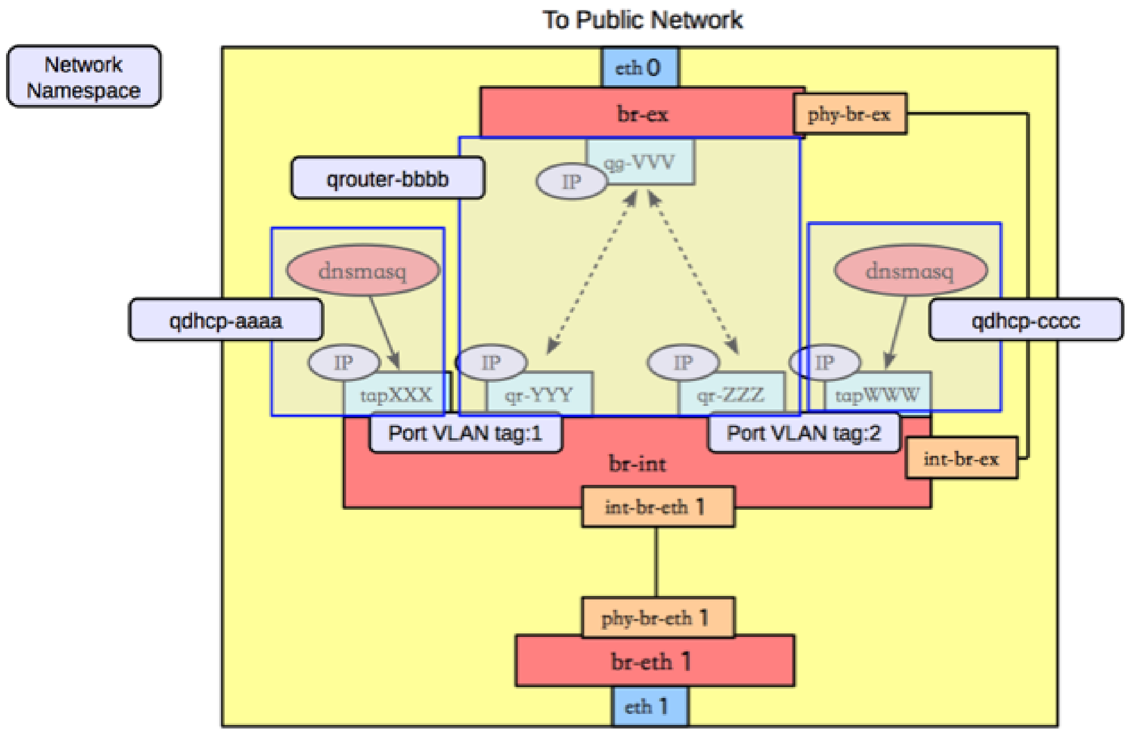
Overlapping subnets and network namespaces

One problem with using the host to implement routing is that one of the Networking subnets might overlap with one of the physical networks that the host uses. For example, if the management network is implemented on `eth2` and also happens to be on the `192.168.101.0/24` subnet, routing problems will occur because the host can't determine whether to send a packet on this subnet to `qr-`YYY or `eth2`. If end users are permitted to create their own logical networks and subnets, you must design the system so that such collisions do not occur.

Networking uses Linux *network namespaces* to prevent collisions between the physical networks on the network host, and the logical networks used by the virtual machines. It also prevents collisions across different logical networks that are not routed to each other, as the following scenario shows.

A network namespace is an isolated environment with its own networking stack. A network namespace has its own network interfaces, routes, and iptables rules. Consider it a chroot jail, except for networking instead of for a file system. LXC (Linux containers) use network namespaces to implement networking virtualization.

Networking creates network namespaces on the network host to avoid subnet collisions.

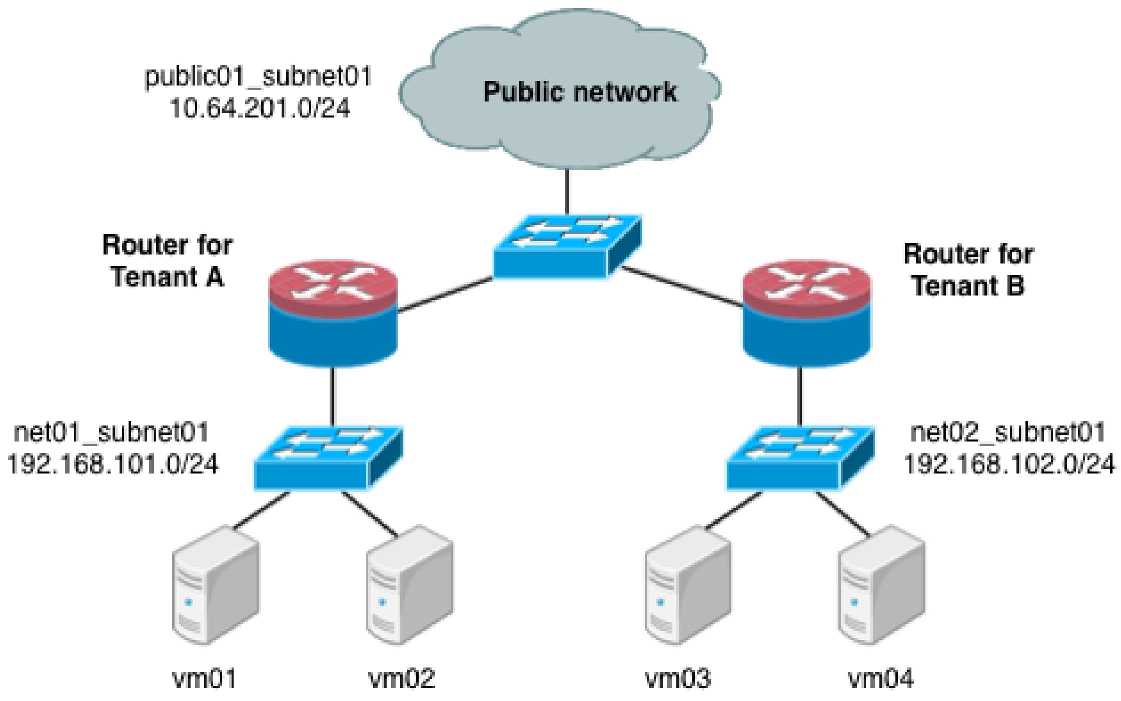


In this example, there are three network namespaces, as shown in the figure above:

- `qdhcp-`*`aaa`*: contains the `tap`*`XXX`* interface and the dnsmasq process that listens on that interface to provide DHCP services for `net01_subnet01`. This allows overlapping IPs between `net01_subnet01` and any other subnets on the network host.
- `qrouter-`*`bbbb`*: contains the `qr-`*`YYY`*, `qr-`*`ZZZ`*, and `qg-`*`VVV`* interfaces, and the corresponding routes. This namespace implements `router01` in our example.
- `qdhcp-`*`ccc`*: contains the `tap`*`WWW`* interface and the dnsmasq process that listens on that interface, to provide DHCP services for `net02_subnet01`. This allows overlapping IPs between `net02_subnet01` and any other subnets on the network host.

Scenario 2: two tenants, two networks, two routers

In this scenario, tenant A and tenant B each have a network with one subnet and one router that connects the tenants to the public Internet.



Under the `service` tenant, define the public network:

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ neutron net-create --tenant-id $tenant public01 \
          --provider:network_type flat \
          --provider:physical_network physnet1 \
          --router:external=True
$ neutron subnet-create --tenant-id $tenant --name public01_subnet01 \
          --gateway 10.64.201.254 public01 10.64.201.0/24 --disable-dhcp
```

Under the `tenantA` user tenant, create the tenant router and set its gateway for the public network.

```
$ tenant=$(keystone tenant-list|awk '/tenantA/ {print $2}')
$ neutron router-create --tenant-id $tenant router01
$ neutron router-gateway-set router01 public01
```

Then, define private network `net01` using VLAN ID 101 on the physical switch, along with its subnet, and connect it to the router.

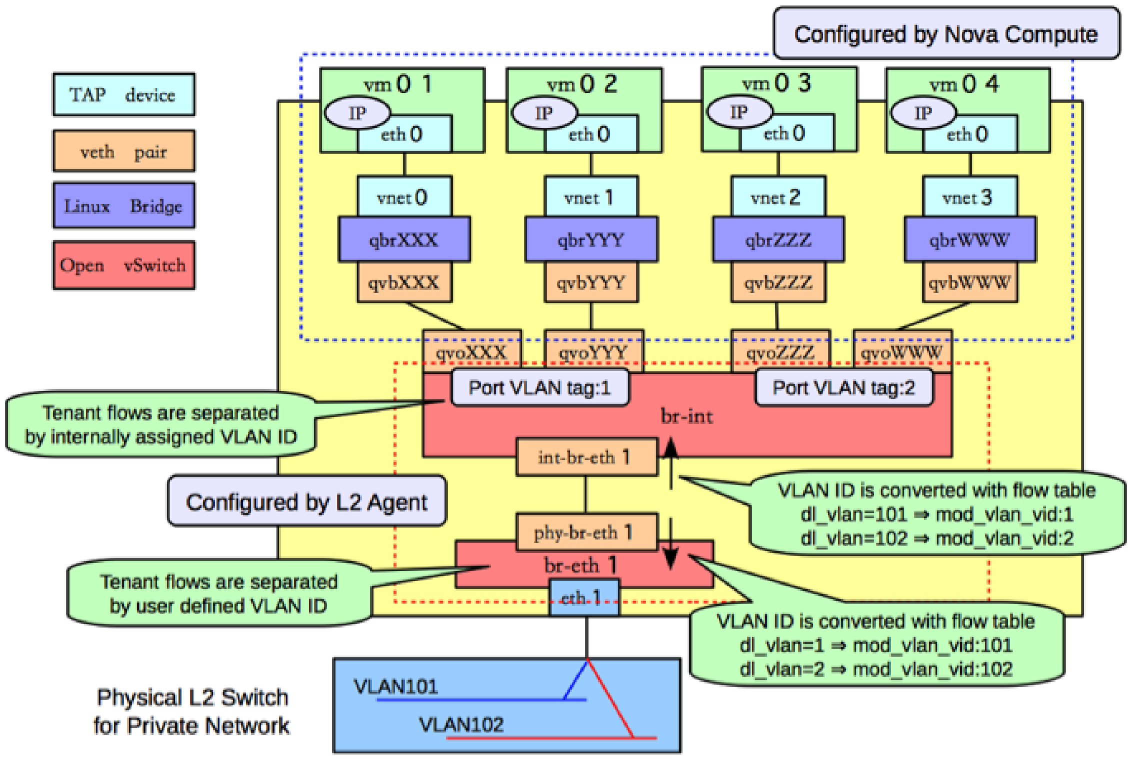
```
$ neutron net-create --tenant-id $tenant net01 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 101
$ neutron subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.
168.101.0/24
$ neutron router-interface-add router01 net01_subnet01
```

Similarly, for `tenantB`, create a router and another network, using VLAN ID 102 on the physical switch:

```
$ tenant=$(keystone tenant-list|awk '/tenantB/ {print $2}')
$ neutron router-create --tenant-id $tenant router02
$ neutron router-gateway-set router02 public01
$ neutron net-create --tenant-id $tenant net02 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 102
$ neutron subnet-create --tenant-id $tenant --name net02_subnet01 net01 192.
168.102.0/24
$ neutron router-interface-add router02 net02_subnet01
```

Scenario 2: Compute host config

The following figure shows how to configure Linux networking devices on the compute host:

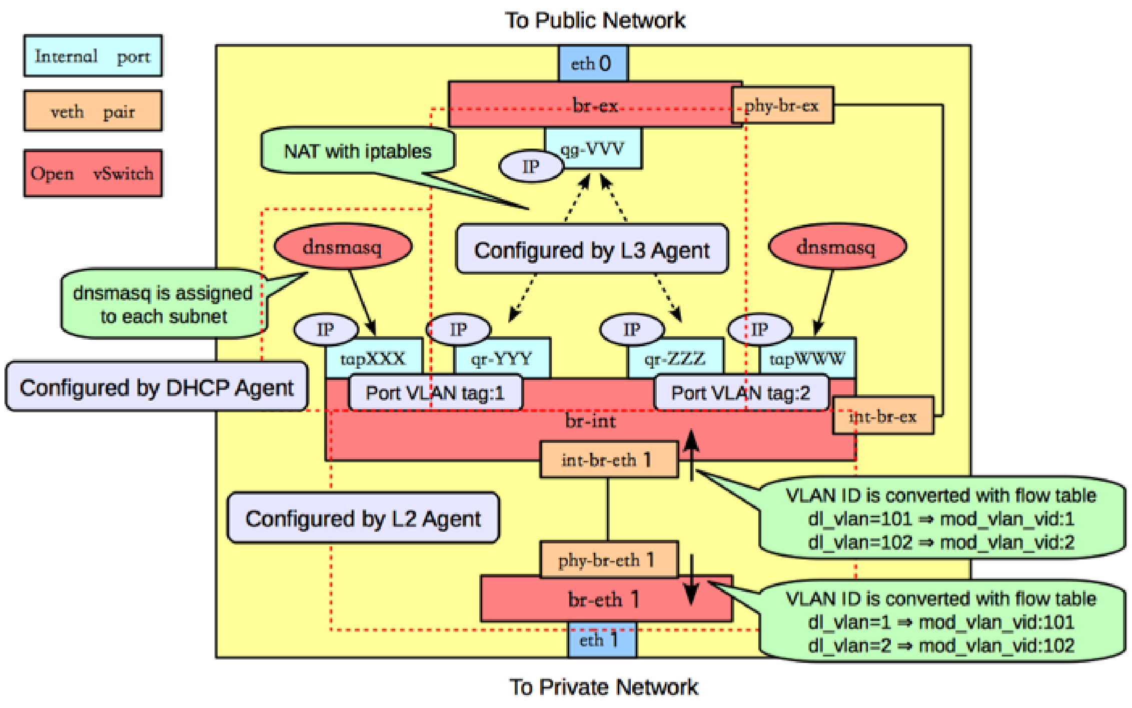

Configured by Nova Compute
TAP device
veth pair
Linux Bridge
Open vSwitch
vm01
vm02
vm03
vm04
IP
eth0
vnet0
vnet1
vnet2
vnet3
qbrXXX
qbrYYY
qbrZZZ
qbrWWW
qvbXXX
qvbYYY
qvbZZZ
qvbWWW
qvoXXX
qvoYYY
qvoZZZ
qvoWWW
Port VLAN tag:1
Port VLAN tag:2
br-int
Tenant flows are separated by internally assigned VLAN ID
int-br-eth1
Configured by L2 Agent
VLAN ID is converted with flow table dl_vlan=101 ⇒ mod_vlan_vid:1 dl_vlan=102 ⇒ mod_vlan_vid:2
phy-br-eth1
br-eth1
eth1
Tenant flows are separated by user defined VLAN ID
VLAN ID is converted with flow table dl_vlan=1 ⇒ mod_vlan_vid:101 dl_vlan=2 ⇒ mod_vlan_vid:102
Physical L2 Switch for Private Network
VLAN101
VLAN102

Note

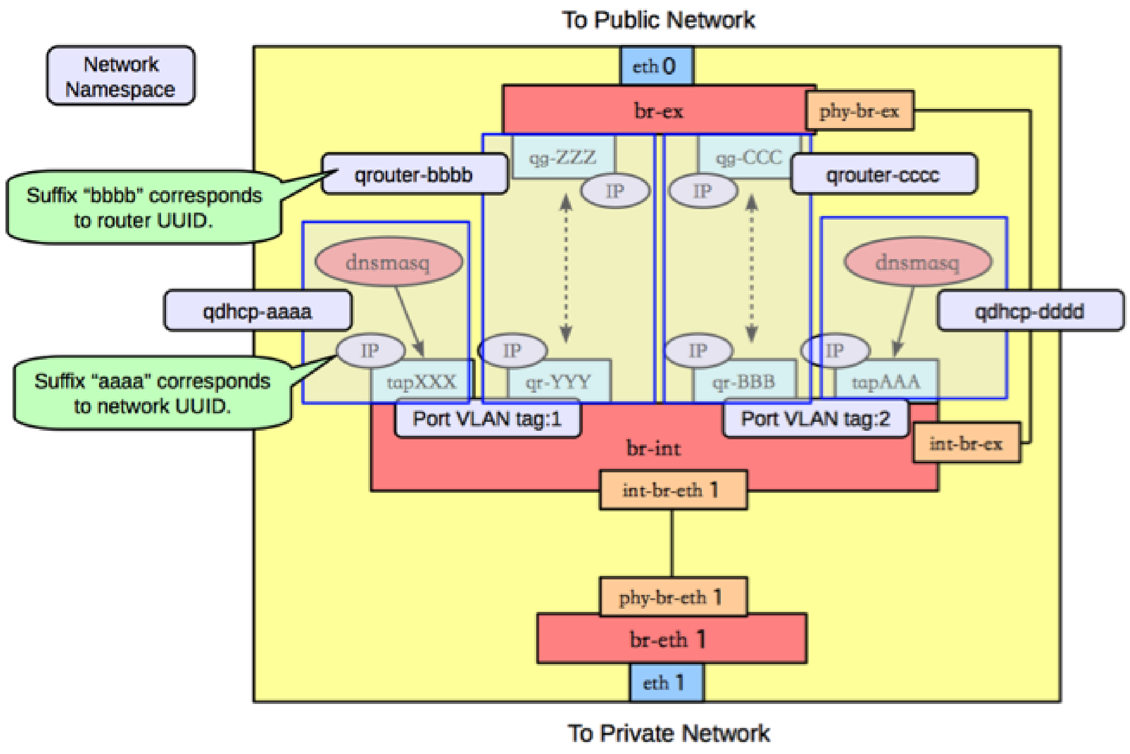
The compute host configuration resembles the configuration in scenario 1. However, in scenario 1, a guest connects to two subnets while in this scenario, the subnets belong to different tenants.

Scenario 2: Network host config

The following figure shows the network devices on the network host for the second scenario.



In this configuration, the network namespaces are organized to isolate the two subnets from each other as shown in the following figure.

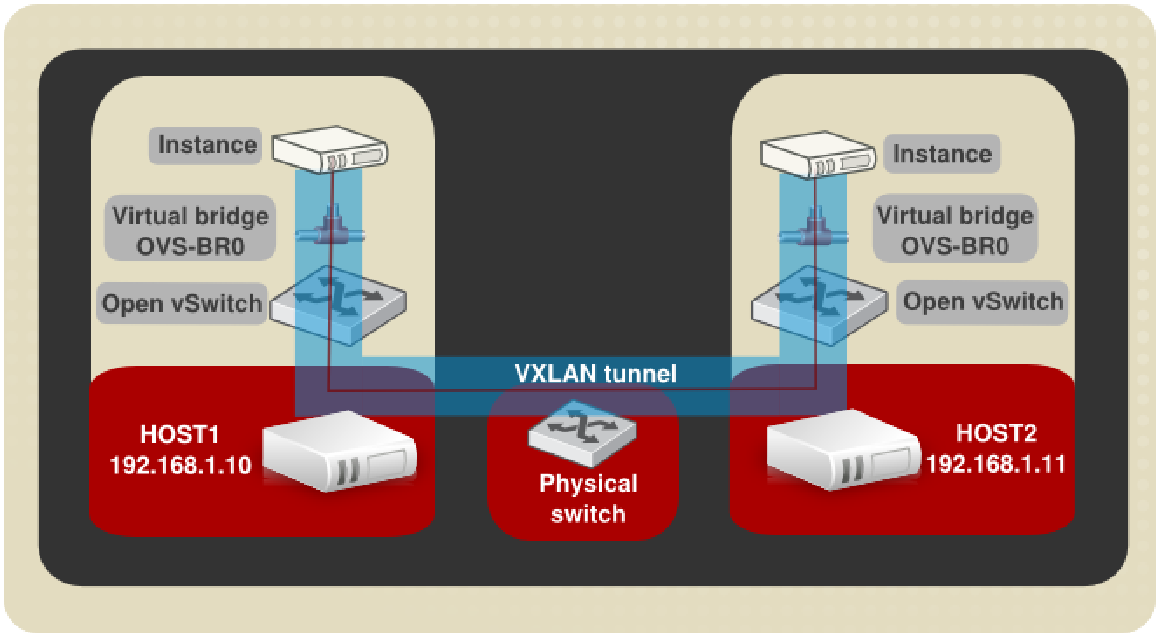


In this scenario, there are four network namespaces (`qhdcp-`*aaa*, `qrouter-`*bbbb*, `qrouter-`*cccc*, and `qhdcp-`*dddd*), instead of three. Since there is no connectivity between the two networks, and so each router is implemented by a separate namespace.

Configure Open vSwitch tunneling

Tunneling encapsulates network traffic between physical Networking hosts and allows VLANs to span multiple physical hosts. Instances communicate as if they share the same layer 2 network. Open vSwitch supports tunneling with the VXLAN and GRE encapsulation protocols.

Figure 7.3. Example VXLAN tunnel



This diagram shows two instances running on separate hosts connected by a VXLAN tunnel. The required physical and virtual components are also illustrated. The following procedure creates a VXLAN or GRE tunnel between two Open vSwitches running on separate Networking hosts:

Procedure 7.16. Example tunnel configuration

1. Create a virtual bridge named OVS-BR0 on each participating host:

```
ovs-vsctl add-br OVS-BR0
```

2. Create a tunnel to link the OVS-BR0 virtual bridges. Run the ovs-vsctl command on HOST1 to create the tunnel and link it to the bridge on HOST2:

   GRE tunnel command:

```
ovs-vsctl add-port OVS-BR0 gre1 -- set Interface gre1 type=gre
 options:remote_ip=192.168.1.11
```

VXLAN tunnel command:

```
ovs-vsctl add-port OVS-BR0 vxlan1 -- set Interface vxlan1 type=vxlan
 options:remote_ip=192.168.1.11
```

3. Run the ovs-vsctl command on HOST1 to create the tunnel and link it to the bridge on HOST2.

   GRE tunnel command:

   ```
   ovs-vsctl add-port OVS-BR0 gre1 -- set Interface gre1 type=gre
    options:remote_ip=192.168.1.10
   ```

   VXLAN tunnel command:

   ```
   ovs-vsctl add-port OVS-BR0 vxlan1 -- set Interface vxlan1 type=vxlan
    options:remote_ip=192.168.1.10
   ```

Successful completion of these steps results in the two instances sharing a layer 2 network.

Linux Bridge

This section describes how the Linux Bridge plug-in implements the Networking abstractions. For information about DHCP and L3 agents, see the section called "Scenario 1: one tenant, two networks, one router" [192].

Configuration

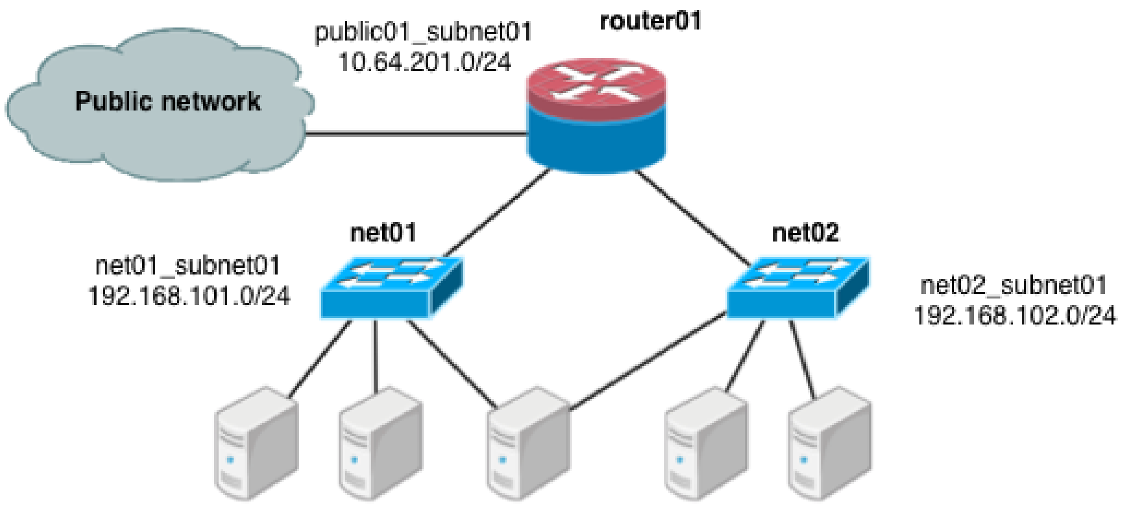
This example uses VLAN isolation on the switches to isolate tenant networks. This configuration labels the physical network associated with the public network as `physnet1`, and the physical network associated with the data network as `physnet2`, which leads to the following configuration options in `linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet2:100:110

[linux_bridge]
physical_interface_mappings = physnet2:eth1
```

Scenario 1: one tenant, two networks, one router

The first scenario has two private networks (`net01`, and `net02`), each with one subnet (`net01_subnet01`: 192.168.101.0/24, `net02_subnet01`, 192.168.102.0/24). Both private networks are attached to a router that contains them to the public network (10.64.201.0/24).



Under the `service` tenant, create the shared router, define the public network, and set it as the default gateway of the router

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ neutron router-create router01
$ neutron net-create --tenant-id $tenant public01 \
          --provider:network_type flat \
          --provider:physical_network physnet1 \
          --router:external=True
$ neutron subnet-create --tenant-id $tenant --name public01_subnet01 \
          --gateway 10.64.201.254 public01 10.64.201.0/24 --disable-dhcp
$ neutron router-gateway-set router01 public01
```

Under the `demo` user tenant, create the private network `net01` and corresponding subnet, and connect it to the `router01` router. Configure it to use VLAN ID 101 on the physical switch.

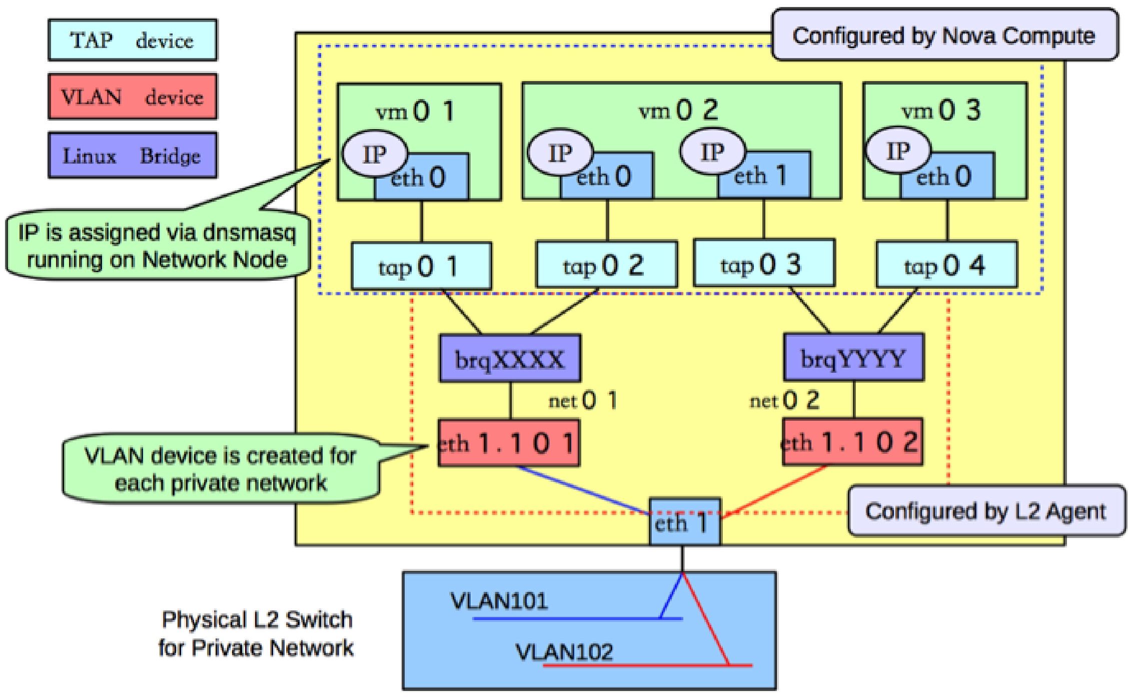
```
$ tenant=$(keystone tenant-list|awk '/demo/ {print $2}'
$ neutron net-create --tenant-id $tenant net01 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 101
$ neutron subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.
168.101.0/24
$ neutron router-interface-add router01 net01_subnet01
```

Similarly, for `net02`, using VLAN ID 102 on the physical switch:

```
$ neutron net-create --tenant-id $tenant net02 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 102
$ neutron subnet-create --tenant-id $tenant --name net02_subnet01 net02 192.
168.102.0/24
$ neutron router-interface-add router01 net02_subnet01
```

Scenario 1: Compute host config

The following figure shows how to configure the various Linux networking devices on the compute host.


TAP device
VLAN device
Linux Bridge
Configured by Nova Compute
vm01
vm02
vm03
IP
eth0
eth1
tap01
tap02
tap03
tap04
IP is assigned via dnsmasq running on Network Node
brqXXXX
brqYYYY
net01
net02
eth1.101
eth1.102
VLAN device is created for each private network
eth1
Configured by L2 Agent
Physical L2 Switch for Private Network
VLAN101
VLAN102

Types of network devices

Note

There are three distinct type of virtual networking devices: TAP devices, VLAN devices, and Linux bridges. For an ethernet frame to travel from `eth0` of virtual machine `vm01`, to the physical network, it must pass through four devices inside of the host: TAP `vnet0`, Linux bridge `brq`*`XXX`*, VLAN `eth1.101)`, and, finally, the physical network interface card `eth1`.

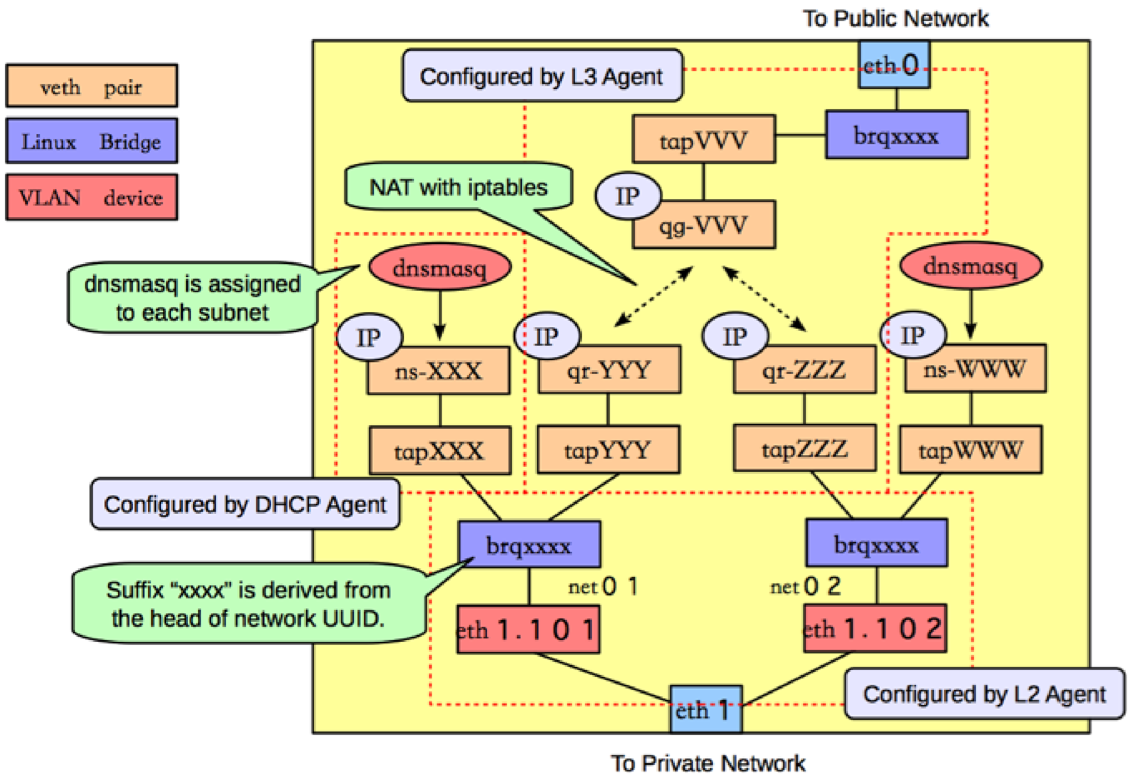
A *TAP device*, such as `vnet0` is how hypervisors such as KVM and Xen implement a virtual network interface card (typically called a VIF or vNIC). An ethernet frame sent to a TAP device is received by the guest operating system.

A *VLAN device* is associated with a VLAN tag attaches to an existing interface device and adds or removes VLAN tags. In the preceding example, VLAN device `eth1.101` is associated with VLAN ID 101 and is attached to interface `eth1`. Packets received from the outside by `eth1` with VLAN tag 101 will be passed to device `eth1.101`, which will then strip the tag. In the other direction, any ethernet frame sent directly to eth1.101 will have VLAN tag 101 added and will be forward to `eth1` for sending out to the network.

A *Linux bridge* behaves like a hub: you can connect multiple (physical or virtual) network interfaces devices to a Linux bridge. Any ethernet frames that come in from one interface attached to the bridge is transmitted to all of the other devices.

Scenario 1: Network host config

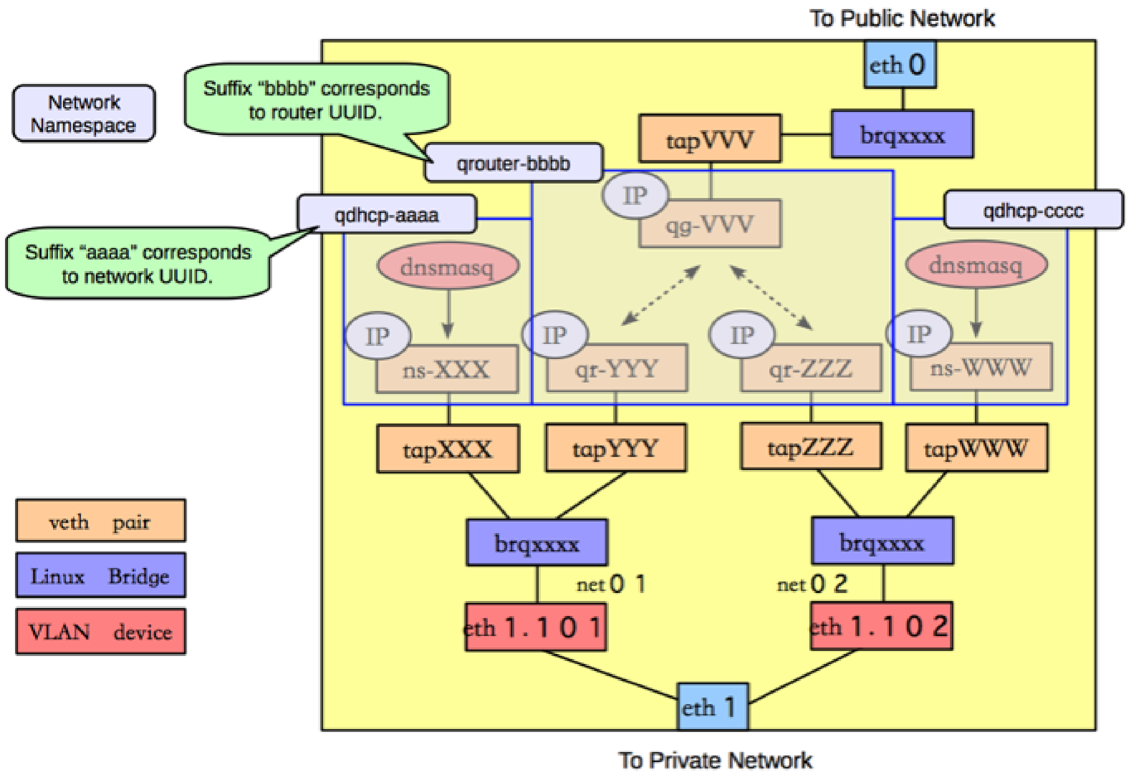
The following figure shows the network devices on the network host.



The following figure shows how the Linux Bridge plug-in uses network namespaces to provide isolation.

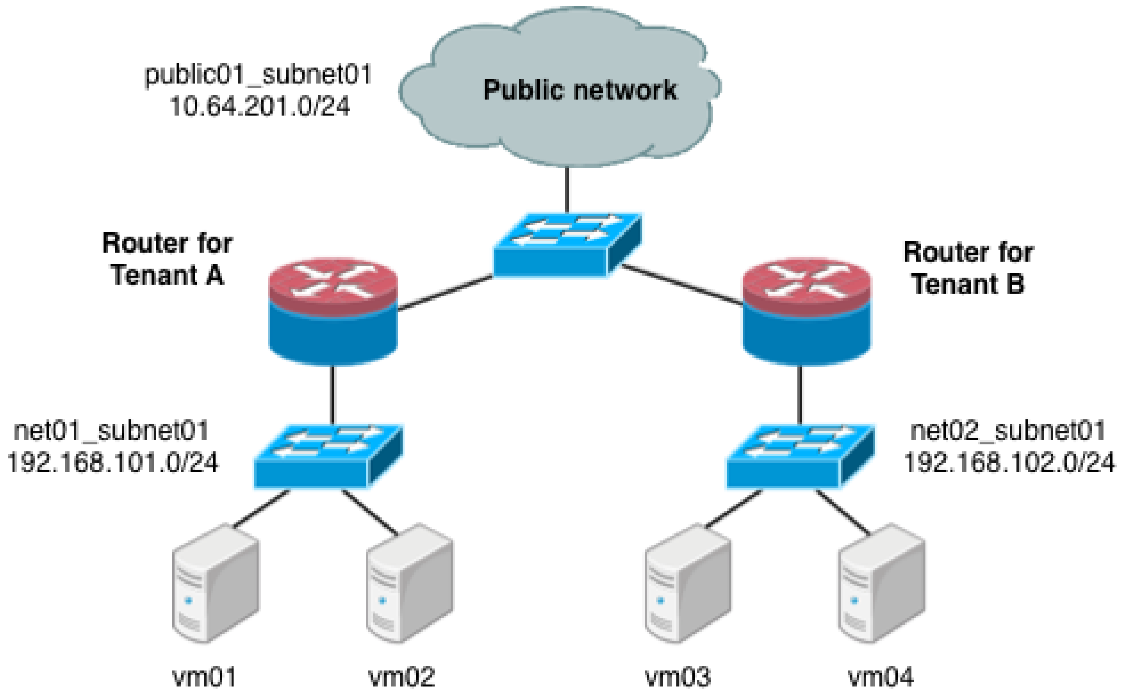
Note

veth pairs form connections between the Linux bridges and the network namespaces.



Scenario 2: two tenants, two networks, two routers

The second scenario has two tenants (A, B). Each tenant has a network with one subnet, and each one has a router that connects them to the public Internet.



Under the `service` tenant, define the public network:

```
$ tenant=$(keystone tenant-list | awk '/service/ {print $2}')
$ neutron net-create --tenant-id $tenant public01 \
          --provider:network_type flat \
          --provider:physical_network physnet1 \
          --router:external=True
$ neutron subnet-create --tenant-id $tenant --name public01_subnet01 \
          --gateway 10.64.201.254 public01 10.64.201.0/24 --disable-dhcp
```

Under the `tenantA` user tenant, create the tenant router and set its gateway for the public network.

```
$ tenant=$(keystone tenant-list|awk '/tenantA/ {print $2}')
$ neutron router-create --tenant-id $tenant router01
$ neutron router-gateway-set router01 public01
```

Then, define private network `net01` using VLAN ID 102 on the physical switch, along with its subnet, and connect it to the router.

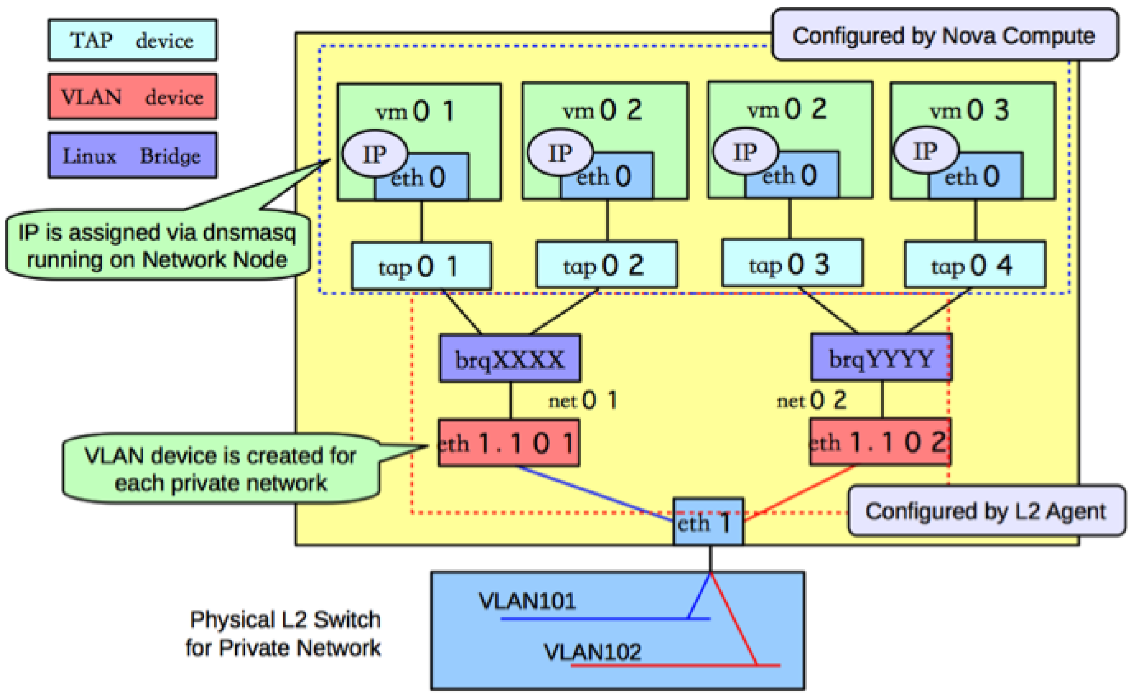
```
$ neutron net-create --tenant-id $tenant net01 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 101
$ neutron subnet-create --tenant-id $tenant --name net01_subnet01 net01 192.
168.101.0/24
$ neutron router-interface-add router01 net01_subnet01
```

Similarly, for `tenantB`, create a router and another network, using VLAN ID 102 on the physical switch:

```
$ tenant=$(keystone tenant-list|awk '/tenantB/ {print $2}')
$ neutron router-create --tenant-id $tenant router02
$ neutron router-gateway-set router02 public01
$ neutron net-create --tenant-id $tenant net02 \
          --provider:network_type vlan \
          --provider:physical_network physnet2 \
          --provider:segmentation_id 102
$ neutron subnet-create --tenant-id $tenant --name net02_subnet01 net01 192.
168.101.0/24
$ neutron router-interface-add router02 net02_subnet01
```

Scenario 2: Compute host config

The following figure shows how the various Linux networking devices would be configured on the compute host under this scenario.

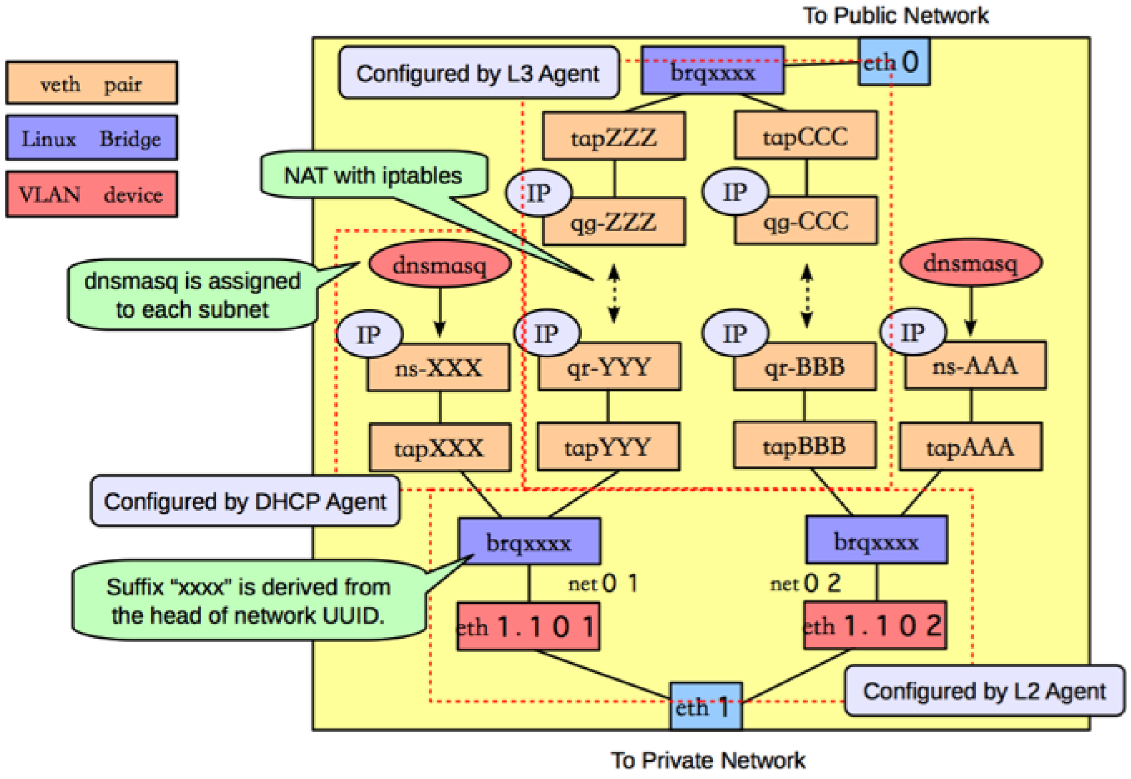

TAP device
VLAN device
Linux Bridge
Configured by Nova Compute
vm01
vm02
vm02
vm03
IP
eth0
tap01
tap02
tap03
tap04
IP is assigned via dnsmasq running on Network Node
brqXXXX
brqYYYY
net01
net02
eth1.101
eth1.102
VLAN device is created for each private network
eth1
Configured by L2 Agent
Physical L2 Switch for Private Network
VLAN101
VLAN102

Note

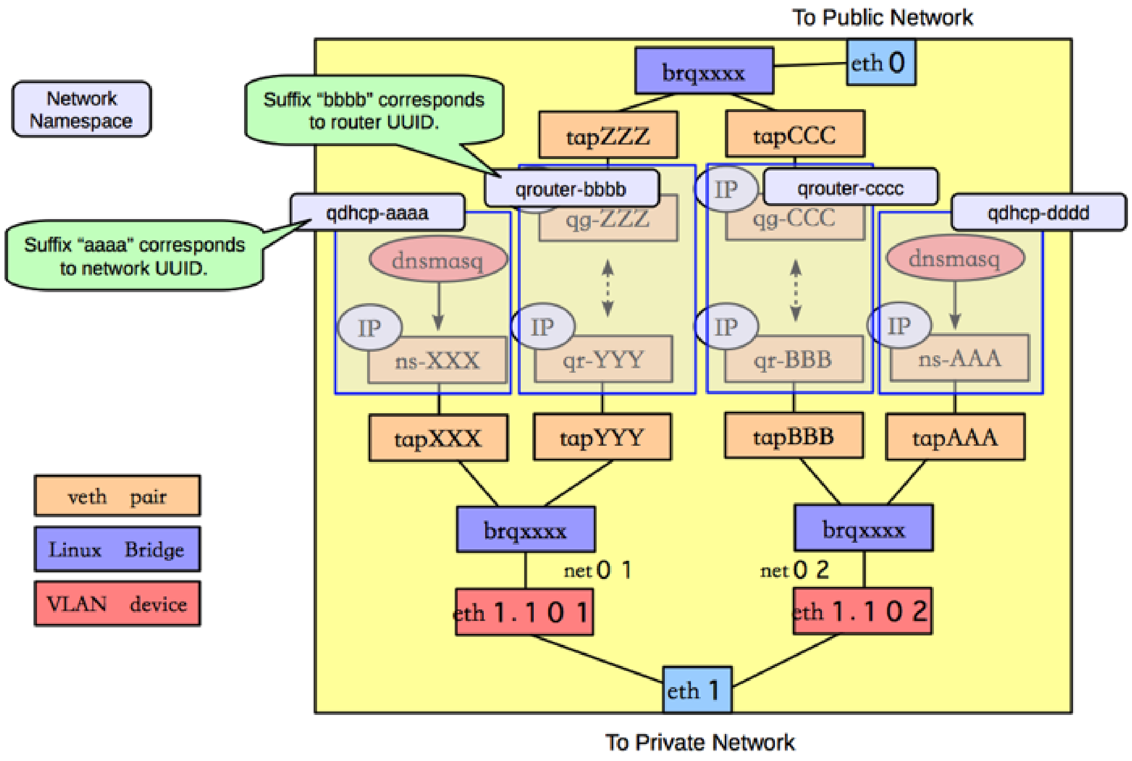
The configuration on the compute host is very similar to the configuration in scenario 1. The only real difference is that scenario 1 had a guest connected to two subnets, and in this scenario the subnets belong to different tenants.

Scenario 2: Network host config

The following figure shows the network devices on the network host for the second scenario.



The main difference between the configuration in this scenario and the previous one is the organization of the network namespaces, in order to provide isolation across the two subnets, as shown in the following figure.



In this scenario, there are four network namespaces (`qhdcp-`*aaa*, `qrouter-`*bbbb*, `qrouter-`*cccc*, and `qhdcp-`*dddd*), instead of three. Each router is implemented by a separate namespace, since there is no connectivity between the two networks.

ML2

The Modular Layer 2 plug-in allows OpenStack Networking to simultaneously utilize the variety of layer 2 networking technologies found in complex real-world data centers. It currently includes drivers for the local, flat, VLAN, GRE and VXLAN network types and works with the existing *Open vSwitch*, *Linux Bridge* , and *HyperV* L2 agents. The *ML2* plug-in can be extended through mechanism drivers, allowing multiple mechanisms to be used simultaneously. This section describes different *ML2* plug-in and agent configurations with different type drivers and mechanism drivers.

Previously, Networking deployments were only able to use the plug-in that had been selected at implementation time. For example, a deployment running the Open vSwitch plug-in was only able to use Open vSwitch exclusively; it wasn't possible to simultaneously run another plug-in such as Linux Bridge. This was found to be a limitation in environments with heterogeneous requirements.

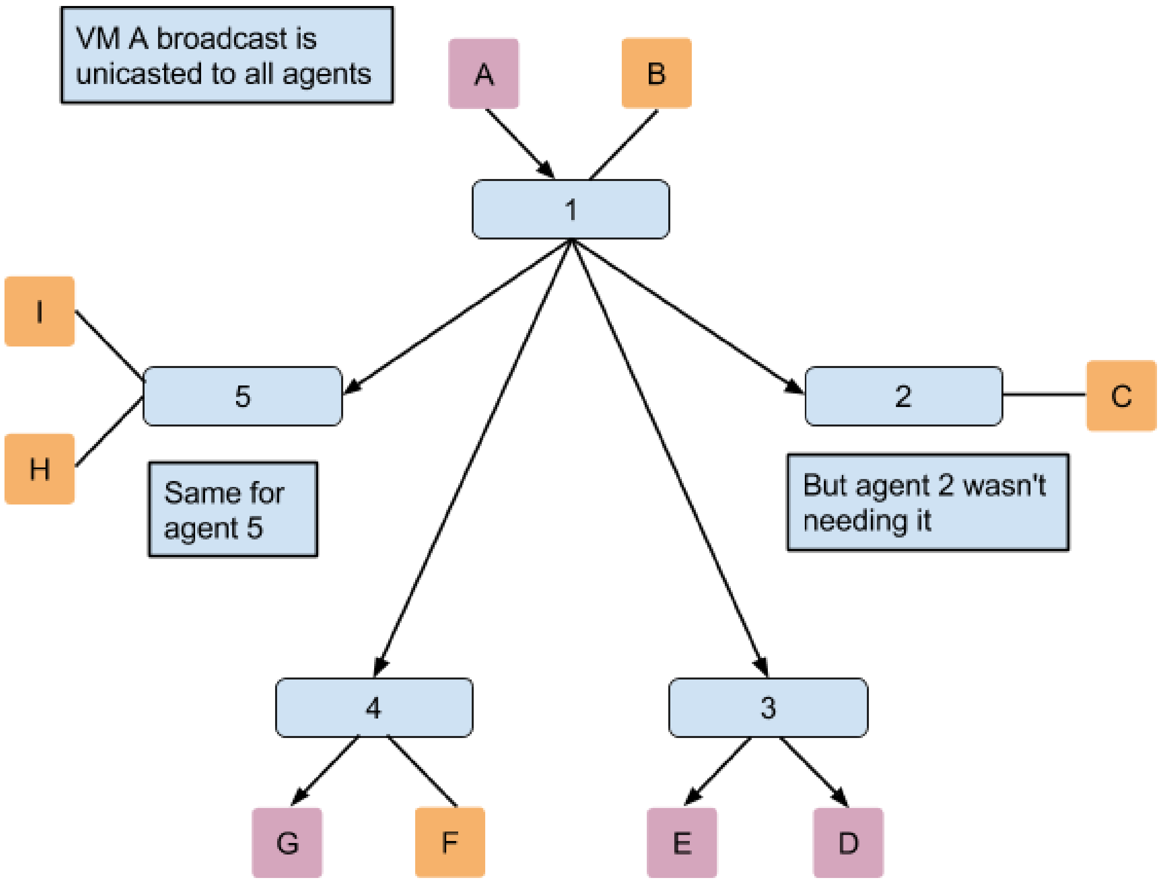
Warning

Disabling a ML2 type driver and re-enabling it later may lead to database inconsistencies if ML2 is reconfigured without support for that type.

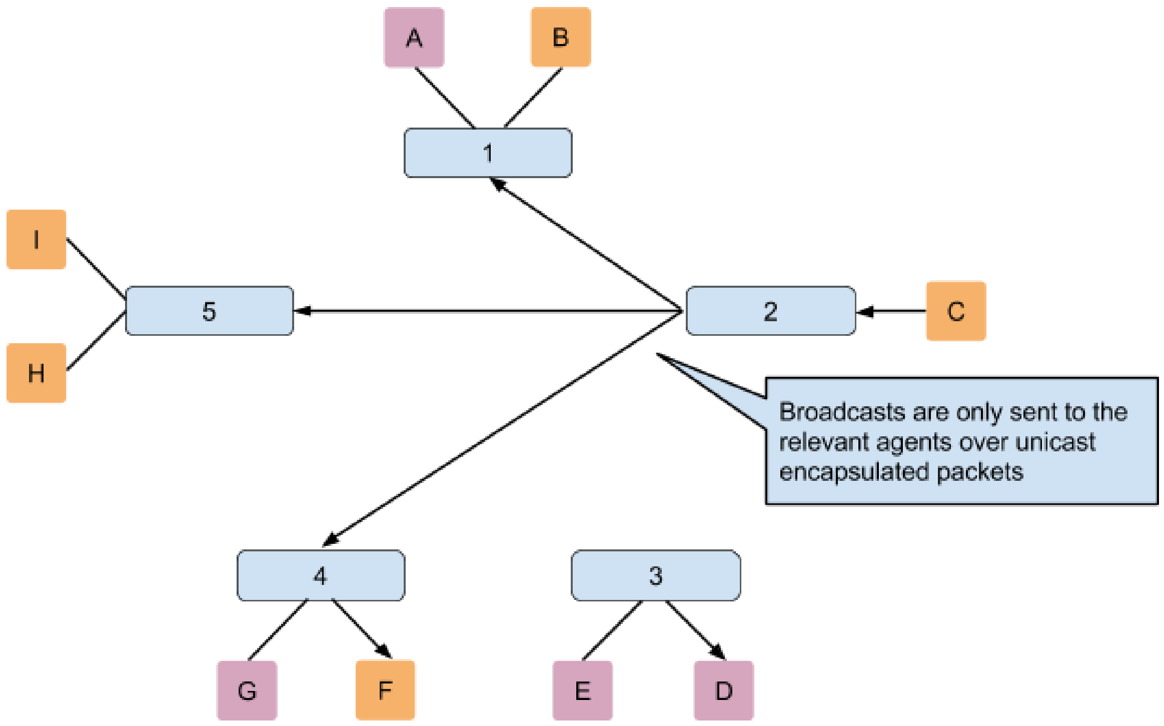
ML2 with L2 population mechanism driver

The L2 Population driver enables broadcast, multicast, and unicast traffic to scale out on large overlay networks. This traffic is sent to the relevant agent via encapsulation as a targeted unicast.

Current *Open vSwitch* and *Linux Bridge* tunneling implementations broadcast to every agent, even if they don't host the corresponding network as illustrated below.


VM A broadcast is unicasted to all agents
A
B
1
I
5
H
Same for agent 5
2
C
But agent 2 wasn't needing it
4
3
G
F
E
D

As broadcast emulation on overlay is costly, it may be better to avoid its use for MAC learning and ARP resolution. This supposes the use of proxy ARP on the agent to answer VM requests, and to populate forwarding table. Currently only the *Linux Bridge* Agent implements an ARP proxy. The prepopulation limits L2 broadcasts in overlay, however it may anyway be necessary to provide broadcast emulation. This is achieved by broadcasting packets via unicast only to the relevant agents as illustrated below.


A
B
1
I
5
2
C
H
Broadcasts are only sent to the relevant agents over unicast encapsulated packets
4
3
G
F
E
D

The partial-mesh is available with the *Open vSwitch* and *Linux Bridge* agents. The following scenarios will use the L2 population mechanism driver with an *Open vSwitch* agent and a *Linux Bridge* agent. Enable the l2 population driver by adding it to the list of mechanism drivers. In addition, a tunneling driver must be selected. Supported options are GRE, VXLAN, or a combination of both. Configuration settings are enabled in `ml2_conf.ini`:

```
[ml2]
type_drivers = local,flat,vlan,gre,vxlan
mechanism_drivers = openvswitch,linuxbridge,l2population
```

Scenario 1: L2 population with Open vSwitch agent

Enable the l2 population extension in the *Open vSwitch* agent, and configure the `local_ip` and `tunnel_types` parameters in the `ml2_conf.ini` file:

```
[ovs]
local_ip = 192.168.1.10

[agent]
tunnel_types = gre,vxlan
l2_population = True
```

Scenario 2: L2 population with *Linux Bridge* agent

Enable the l2 population extension on the *Linux Bridge* agent. Enable VXLAN and configure the local_ip parameter in `ml2_conf.ini`.

```
[vxlan]
enable_vxlan = True
local_ip = 192.168.1.10
l2_population = True
```

Enable security group API

Since the ML2 plug-in can concurrently support different L2 agents (or other mechanisms) with different configuration files, the actual `firewall_driver` value in the `ml2_conf.ini` file does not matter in the server, but `firewall_driver` must be set to a non-default value in the ml2 configuration to enable the securitygroup extension. To enable securitygroup API, edit the `ml2_conf.ini` file:

```
[securitygroup]
firewall_driver = dummy
```

Each L2 agent configuration file (such as `ovs_neutron_plugin.ini` or `linuxbridge_conf.ini`) should contain the appropriate `firewall_driver` value for that agent. To disable securitygroup API, edit the`ml2_conf.ini` file:

```
[securitygroup]
firewall_driver = neutron.agent.firewall.NoopFirewallDriver
```

Also, each L2 agent configuration file (such as `ovs_neutron_plugin.ini` or `linuxbridge_conf.ini`) should contain this value in `firewall_driver` parameter for that agent.

Advanced configuration options

This section describes advanced configuration options for various system components. For example, configuration options where the default works but that the user wants to customize options. After installing from packages, $NEUTRON_CONF_DIR is `/etc/neutron`.

OpenStack Networking server with plug-in

This is the web server that runs the OpenStack Networking API Web Server. It is responsible for loading a plug-in and passing the API calls to the plug-in for processing. The neutron-server should receive one of more configuration files as it its input, for example:

```
neutron-server --config-file <neutron config> --config-file <plugin config>
```

The neutron config contains the common neutron configuration parameters. The plug-in config contains the plug-in specific flags. The plug-in that is run on the service is loaded through the *`core_plugin`* configuration parameter. In some cases a plug-in might have an agent that performs the actual networking.

Most plug-ins require a SQL database. After you install and start the database server, set a password for the root account and delete the anonymous accounts:

```
$> mysql -u root
mysql> update mysql.user set password = password('iamroot') where user =
 'root';
mysql> delete from mysql.user where user = '';
```

Create a database and user account specifically for plug-in:

```
mysql> create database <database-name>;
mysql> create user '<user-name>'@'localhost' identified by '<user-name>';
mysql> create user '<user-name>'@'%' identified by '<user-name>';
mysql> grant all on <database-name>.* to '<user-name>'@'%';
```

Once the above is done you can update the settings in the relevant plug-in configuration files. The plug-in specific configuration files can be found at $NEUTRON_CONF_DIR/plugins.

Some plug-ins have a L2 agent that performs the actual networking. That is, the agent will attach the virtual machine NIC to the OpenStack Networking network. Each node should have an L2 agent running on it. Note that the agent receives the following input parameters:

```
neutron-plugin-agent --config-file <neutron config> --config-file <plugin
 config>
```

Two things need to be done prior to working with the plug-in:

1. Ensure that the core plug-in is updated.

2. Ensure that the database connection is correctly set.

The following table contains examples for these settings. Some Linux packages might provide installation utilities that configure these.

Table 7.10. Settings

Parameter	Value
Open vSwitch	
core_plugin ($NEUTRON_CONF_DIR/ neutron.conf)	neutron.plugins.openvswitch.ovs_neutron_plugin.OVSNeutronPluginV2
connection (in the plugin configuration file, section `[database]`)	mysql://<username>:<password>@localhost/ovs_neutron?charset=utf8
Plug-in Configuration File	$NEUTRON_CONF_DIR/plugins/openvswitch/ovs_neutron_plugin.ini
Agent	neutron-openvswitch-agent
Linux Bridge	
core_plugin ($NEUTRON_CONF_DIR/ neutron.conf)	neutron.plugins.linuxbridge.lb_neutron_plugin.LinuxBridgePluginV2
connection (in the plug-in configuration file, section `[database]`)	mysql://<username>:<password>@localhost/neutron_linux_bridge? charset=utf8
Plug-in Configuration File	$NEUTRON_CONF_DIR/plugins/linuxbridge/linuxbridge_conf.ini
Agent	neutron-linuxbridge-agent

All plug-in configuration files options can be found in the Appendix - Configuration File Options.

DHCP agent

There is an option to run a DHCP server that will allocate IP addresses to virtual machines running on the network. When a subnet is created, by default, the subnet has DHCP enabled.

The node that runs the DHCP agent should run:

```
neutron-dhcp-agent --config-file <neutron config>
--config-file <dhcp config>
```

Currently the DHCP agent uses dnsmasq to perform that static address assignment.

A driver needs to be configured that matches the plug-in running on the service.

Table 7.11. Basic settings

Parameter	Value
Open vSwitch	
interface_driver ($NEUTRON_CONF_DIR/dhcp_agent.ini)	neutron.agent.linux.interface.OVSInterfaceDriver
Linux Bridge	
interface_driver ($NEUTRON_CONF_DIR/dhcp_agent.ini)	neutron.agent.linux.interface.BridgeInterfaceDriver

Namespace

By default the DHCP agent makes use of Linux network namespaces in order to support overlapping IP addresses. Requirements for network namespaces support are described in the Limitations section.

If the Linux installation does not support network namespace, you must disable using network namespace in the DHCP agent config file (The default value of use_namespaces is True).

```
use_namespaces = False
```

L3 Agent

There is an option to run a L3 agent that will give enable layer 3 forwarding and floating IP support. The node that runs the L3 agent should run:

```
neutron-l3-agent --config-file <neutron config>
--config-file <l3 config>
```

A driver needs to be configured that matches the plug-in running on the service. The driver is used to create the routing interface.

Table 7.12. Basic settings

Parameter	Value
Open vSwitch	
interface_driver ($NEUTRON_CONF_DIR/l3_agent.ini)	neutron.agent.linux.interface.OVSInterfaceDriver
external_network_bridge ($NEUTRON_CONF_DIR/ l3_agent.ini)	br-ex
Linux Bridge	
interface_driver ($NEUTRON_CONF_DIR/l3_agent.ini)	neutron.agent.linux.interface.BridgeInterfaceDriver
external_network_bridge ($NEUTRON_CONF_DIR/ l3_agent.ini)	This field must be empty (or the bridge name for the external network).

The L3 agent communicates with the OpenStack Networking server via the OpenStack Networking API, so the following configuration is required:

1. OpenStack Identity authentication:

   ```
   auth_url="$KEYSTONE_SERVICE_PROTOCOL://$KEYSTONE_AUTH_HOST:
   $KEYSTONE_AUTH_PORT/v2.0"
   ```

   For example,

   ```
   http://10.56.51.210:5000/v2.0
   ```

2. Admin user details:

   ```
   admin_tenant_name $SERVICE_TENANT_NAME
   admin_user $Q_ADMIN_USERNAME
   admin_password $SERVICE_PASSWORD
   ```

Namespace

By default the L3 agent makes use of Linux network namespaces in order to support overlapping IP addresses. Requirements for network namespaces support are described in the Limitation section.

If the Linux installation does not support network namespace, you must disable using network namespace in the L3 agent config file (The default value of use_namespaces is True).

```
use_namespaces = False
```

When use_namespaces is set to False, only one router ID can be supported per node. This must be configured via the configuration variable *router_id*.

```
# If use_namespaces is set to False then the agent can only configure one
 router.
# This is done by setting the specific router_id.
router_id = 1064ad16-36b7-4c2f-86f0-daa2bcbd6b2a
```

To configure it, you need to run the OpenStack Networking service and create a router, and then set an ID of the router created to *router_id* in the L3 agent configuration file.

```
$ neutron router-create myrouter1
Created a new router:
+-----------------------+--------------------------------------+
| Field                 | Value                                |
+-----------------------+--------------------------------------+
| admin_state_up        | True                                 |
| external_gateway_info |                                      |
| id                    | 338d42d7-b22e-42c5-9df6-f3674768fe75 |
| name                  | myrouter1                            |
| status                | ACTIVE                               |
| tenant_id             | 0c236f65baa04e6f9b4236b996555d56     |
+-----------------------+--------------------------------------+
```

Multiple floating IP pools

The L3 API in OpenStack Networking supports multiple floating IP pools. In OpenStack Networking, a floating IP pool is represented as an external network and a floating IP is allocated from a subnet associated with the external network. Since each L3 agent can be associated with at most one external network, we need to invoke multiple L3 agent to define multiple floating IP pools. **'gateway_external_network_id'** in L3 agent configuration file indicates the external network that the L3 agent handles. You can run multiple L3 agent instances on one host.

In addition, when you run multiple L3 agents, make sure that **handle_internal_only_routers** is set to **True** only for one L3 agent in an OpenStack Networking deployment and set to **False** for all other L3 agents. Since the default value of this parameter is True, you need to configure it carefully.

Before starting L3 agents, you need to create routers and external networks, then update the configuration files with UUID of external networks and start L3 agents.

For the first agent, invoke it with the following l3_agent.ini where handle_internal_only_routers is True.

```
handle_internal_only_routers = True
gateway_external_network_id = 2118b11c-011e-4fa5-a6f1-2ca34d372c35
external_network_bridge = br-ex
```

```
python /opt/stack/neutron/bin/neutron-l3-agent
 --config-file /etc/neutron/neutron.conf
 --config-file=/etc/neutron/l3_agent.ini
```

For the second (or later) agent, invoke it with the following l3_agent.ini where handle_internal_only_routers is False.

```
handle_internal_only_routers = False
gateway_external_network_id = e828e54c-850a-4e74-80a8-8b79c6a285d8
external_network_bridge = br-ex-2
```

L3 Metering Agent

There is an option to run a L3 metering agent that will enable layer 3 traffic metering. In general case the metering agent should be launched on all nodes that run the L3 agent:

```
neutron-metering-agent --config-file <neutron config>
--config-file <l3 metering config>
```

A driver needs to be configured that matches the plug-in running on the service. The driver is used to add metering to the routing interface.

Table 7.13. Basic settings

Parameter	Value
Open vSwitch	
interface_driver ($NEUTRON_CONF_DIR/metering_agent.ini)	neutron.agent.linux.interface.OVSInterfaceDriver
Linux Bridge	
interface_driver ($NEUTRON_CONF_DIR/metering_agent.ini)	neutron.agent.linux.interface.BridgeInterfaceDriver

Namespace

The metering agent and the L3 agent have to have the same configuration regarding to the network namespaces setting.

Note

If the Linux installation does not support network namespace, you must disable using network namespace in the L3 metering config file (The default value of `use_namespaces` is `True`).

```
use_namespaces = False
```

L3 metering driver

A driver which implements the metering abstraction needs to be configured. Currently there is only one implementation which is based on iptables.

```
driver = neutron.services.metering.drivers.iptables.iptables_driver.
IptablesMeteringDriver
```

L3 metering service driver

To enable L3 metering you have to be sure to set the following parameter in `neutron.conf` on the host that runs `neutron-server`:

```
service_plugins = neutron.services.metering.metering_plugin.MeteringPlugin
```

Limitations

- *No equivalent for nova-network –multi_host flag:* Nova-network has a model where the L3, NAT, and DHCP processing happen on the compute node itself, rather than a dedicated networking node. OpenStack Networking now support running multiple l3-agent and dhcp-agents with load being split across those agents, but the tight coupling of that scheduling with the location of the VM is not supported in Icehouse. The Juno release is expected to include an exact replacement for the –multi_host flag in nova-network.

- *Linux network namespace required on nodes running `neutron-l3-agent` or `neutron-dhcp-agent` if overlapping IPs are in use:* . In order to support overlapping IP addresses, the OpenStack Networking DHCP and L3 agents use Linux network namespaces by default. The hosts running these processes must support network namespaces. To support network namespaces, the following are required:

  - Linux kernel 2.6.24 or newer (with CONFIG_NET_NS=y in kernel configuration) and

  - iproute2 utilities ('ip' command) version 3.1.0 (aka 20111117) or newer

  To check whether your host supports namespaces try running the following as root:

  ```
  # ip netns add test-ns
  # ip netns exec test-ns ifconfig
  ```

  If the preceding commands do not produce errors, your platform is likely sufficient to use the dhcp-agent or l3-agent with namespace. In our experience, Ubuntu 12.04 or later support namespaces as does Fedora 17 and new, but some older RHEL platforms do not by default. It may be possible to upgrade the iproute2 package on a platform that does not support namespaces by default.

  If you need to disable namespaces, make sure the `neutron.conf` used by neutron-server has the following setting:

  ```
  allow_overlapping_ips=False
  ```

  and that the dhcp_agent.ini and l3_agent.ini have the following setting:

  ```
  use_namespaces=False
  ```

Note

If the host does not support namespaces then the `neutron-l3-agent` and `neutron-dhcp-agent` should be run on different hosts. This is due to the fact that there is no isolation between the IP addresses created by the L3 agent and by the DHCP agent. By manipulating the routing the user can ensure that these networks have access to one another.

If you run both L3 and DHCP services on the same node, you should enable namespaces to avoid conflicts with routes:

```
use_namespaces=True
```

- *No IPv6 support for L3 agent:* The `neutron-l3-agent`, used by many plug-ins to implement L3 forwarding, supports only IPv4 forwarding. Currently, there are no errors provided if you configure IPv6 addresses via the API.
- *ZeroMQ support is experimental*: Some agents, including `neutron-dhcp-agent`, `neutron-openvswitch-agent`, and `neutron-linuxbridge-agent` use RPC to communicate. ZeroMQ is an available option in the configuration file, but has not been tested and should be considered experimental. In particular, issues might occur with ZeroMQ and the dhcp agent.
- *MetaPlugin is experimental*: This release includes a MetaPlugin that is intended to support multiple plug-ins at the same time for different API requests, based on the content of those API requests. The core team has not thoroughly reviewed or tested this functionality. Consider this functionality to be experimental until further validation is performed.

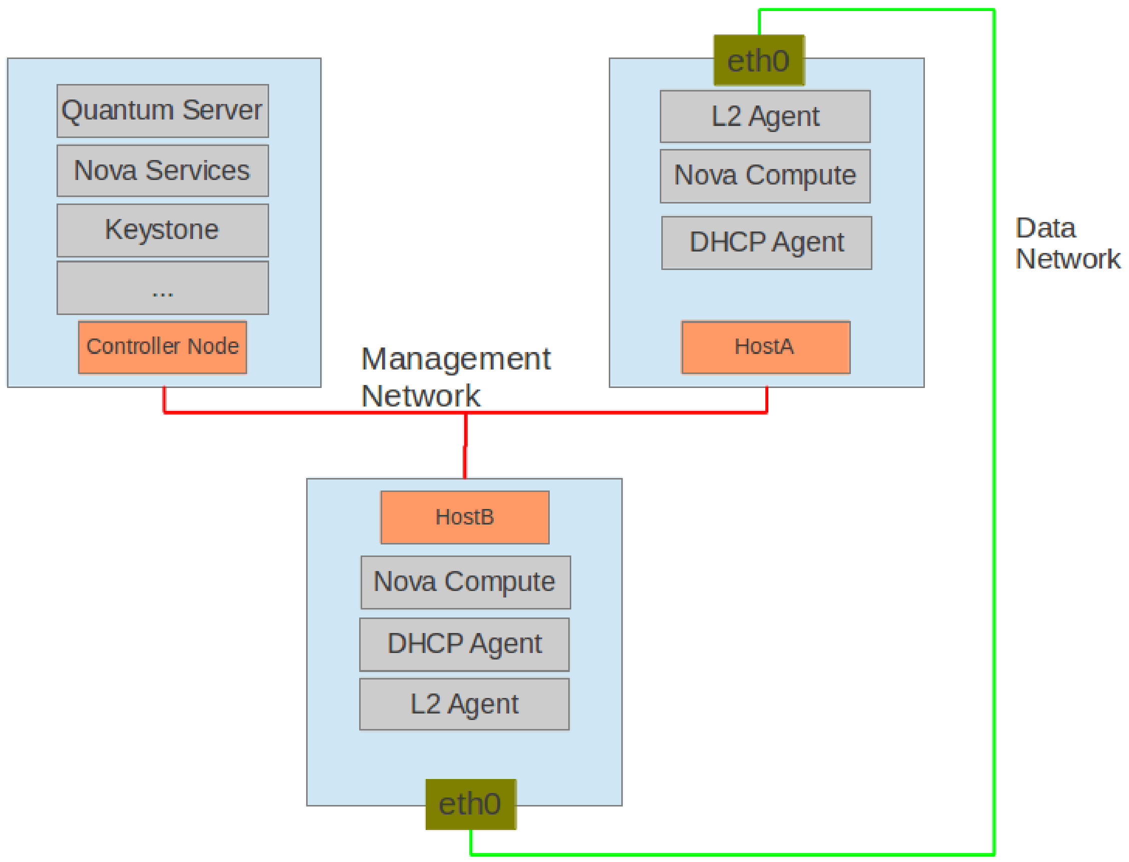
Scalable and highly available DHCP agents

This section describes how to use the agent management (alias agent) and scheduler (alias agent_scheduler) extensions for DHCP agents scalability and HA.

Note

Use the **neutron ext-list** client command to check if these extensions are enabled:

```
$ neutron ext-list -c name -c alias
+-----------------+--------------------------+
| alias           | name                     |
+-----------------+--------------------------+
| agent_scheduler | Agent Schedulers         |
| binding         | Port Binding             |
| quotas          | Quota management support |
| agent           | agent                    |
| provider        | Provider Network         |
| router          | Neutron L3 Router        |
| lbaas           | LoadBalancing service    |
| extraroute      | Neutron Extra Route      |
+-----------------+--------------------------+
```



There will be three hosts in the setup.

Table 7.14. Hosts for demo

Host	Description
OpenStack Controller host - controlnode	Runs the Neutron, Keystone, and Nova services that are required to deploy VMs. The node must have at least one network interface that is connected to the Management Network. Note that `nova-network` should not be running because it is replaced by Neutron.
HostA	Runs Nova compute, the Neutron L2 agent and DHCP agent
HostB	Same as HostA

Configuration

Procedure 7.17. controlnode: Neutron Server

1. Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
core_plugin = neutron.plugins.linuxbridge.lb_neutron_plugin.
LinuxBridgePluginV2
rabbit_host = controlnode
allow_overlapping_ips = True
host = controlnode
agent_down_time = 5
```

2. Update the plug-in configuration file `/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

Procedure 7.18. HostA and HostB: L2 Agent

1. Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
rabbit_host = controlnode
rabbit_password = openstack
# host = HostB on hostb
host = HostA
```

2. Update the plug-in configuration file `/etc/neutron/plugins/linuxbridge/linuxbridge_conf.ini`:

```
[vlans]
tenant_network_type = vlan
network_vlan_ranges = physnet1:1000:2999
[database]
connection = mysql://root:root@127.0.0.1:3306/neutron_linux_bridge
retry_interval = 2
[linux_bridge]
physical_interface_mappings = physnet1:eth0
```

3. Update the nova configuration file `/etc/nova/nova.conf`:

```
[DEFAULT]
network_api_class=nova.network.neutronv2.api.API

neutron_admin_username=neutron
neutron_admin_password=servicepassword
neutron_admin_auth_url=http://controlnode:35357/v2.0/
neutron_auth_strategy=keystone
neutron_admin_tenant_name=servicetenant
neutron_url=http://100.1.1.10:9696/
firewall_driver=nova.virt.firewall.NoopFirewallDriver
```

Procedure 7.19. HostA and HostB: DHCP Agent

- Update the DHCP configuration file `/etc/neutron/dhcp_agent.ini`:

```
[DEFAULT]
interface_driver = neutron.agent.linux.interface.BridgeInterfaceDriver
```

Commands in agent management and scheduler extensions

The following commands require the tenant running the command to have an admin role.

Note

Ensure that the following environment variables are set. These are used by the various clients to access Keystone.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controlnode:5000/v2.0/
```

Procedure 7.20. Settings

- To experiment, you need VMs and a neutron network:

```
$ nova list
+--------------------------------------+-----------+--------
+---------------+
| ID                                   | Name      | Status | Networks
  |
+--------------------------------------+-----------+--------
+---------------+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=10.0.1.
3 |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=10.0.1.
4 |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=10.0.1.
5 |
+--------------------------------------+-----------+--------
+---------------+

$ neutron net-list
+--------------------------------------+------
+--------------------------------------+
| id                                   | name | subnets
         |
+--------------------------------------+------
+--------------------------------------+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-
d5cf646db9d1 |
+--------------------------------------+------
+--------------------------------------+
```

Procedure 7.21. Manage agents in neutron deployment

Every agent which supports these extensions will register itself with the neutron server when it starts up.

1. List all agents:

```
$ neutron agent-list
```

```
+--------------------------------------+--------------------+-------
+-------+----------------+
| id                                   | agent_type         | host  |
 alive | admin_state_up |
+--------------------------------------+--------------------+-------
+-------+----------------+
| 1b69828d-6a9b-4826-87cd-1757f0e27f31 | Linux bridge agent | HostA | :-)
  | True           |
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | DHCP agent         | HostA | :-)
  | True           |
| ed96b856-ae0f-4d75-bb28-40a47ffd7695 | Linux bridge agent | HostB | :-)
  | True           |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent         | HostB | :-)
  | True           |
+--------------------------------------+--------------------+-------
+-------+----------------+
```

The output shows information for four agents. The `alive` field shows `:-)` if the agent reported its state within the period defined by the `agent_down_time` option in the `neutron.conf` file. Otherwise the `alive` is `xxx`.

2. List the DHCP agents that host a specified network

   In some deployments, one DHCP agent is not enough to hold all network data. In addition, you must have a backup for it even when the deployment is small. The same network can be assigned to more than one DHCP agent and one DHCP agent can host more than one network.

   List DHCP agents that host a specified network:

```
$ neutron dhcp-agent-list-hosting-net net1
+--------------------------------------+-------+----------------+-------+
| id                                   | host  | admin_state_up | alive |
+--------------------------------------+-------+----------------+-------+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True           | :-)   |
+--------------------------------------+-------+----------------+-------+
```

3. List the networks hosted by a given DHCP agent.

   This command is to show which networks a given dhcp agent is managing.

```
$ neutron net-list-on-dhcp-agent a0c1c21c-d4f4-4577-9ec7-908f2d48622d
+--------------------------------------+------
+---------------------------------------------------+
| id                                   | name | subnets
                        |
+--------------------------------------+------
+---------------------------------------------------+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-
d5cf646db9d1  10.0.1.0/24 |
+--------------------------------------+------
+---------------------------------------------------+
```

4. Show agent details.

   The **agent-list** command shows details for a specified agent:

```
$ neutron agent-show  a0c1c21c-d4f4-4577-9ec7-908f2d48622d
+---------------------
+-----------------------------------------------------------+
| Field               | Value
       |
+---------------------
+-----------------------------------------------------------+
| admin_state_up      | True
       |
| agent_type          | DHCP agent
       |
| alive               | False
       |
| binary              | neutron-dhcp-agent
       |
| configurations      | {
       |
|                     |      "subnets": 1,
       |
|                     |      "use_namespaces": true,
       |
|                     |      "dhcp_driver": "neutron.agent.linux.dhcp.
Dnsmasq",  |
|                     |      "networks": 1,
       |
|                     |      "dhcp_lease_time": 120,
       |
|                     |      "ports": 3
       |
|                     | }
       |
| created_at          | 2013-03-16T01:16:18.000000
       |
| description         |
       |
| heartbeat_timestamp | 2013-03-17T01:37:22.000000
       |
| host                | HostA
       |
| id                  | 58f4ce07-6789-4bb3-aa42-ed3779db2b03
       |
| started_at          | 2013-03-16T06:48:39.000000
       |
| topic               | dhcp_agent
       |
+---------------------
+-----------------------------------------------------------+
```

In this output, `heartbeat_timestamp` is the time on the neutron server. You do not need to synchronize all agents to this time for this extension to run correctly. `configurations` describes the static configuration for the agent or run time data. This agent is a DHCP agent and it hosts one network, one subnet, and three ports.

Different types of agents show different details. The following output shows information for a Linux bridge agent:

```
$ neutron agent-show ed96b856-ae0f-4d75-bb28-40a47ffd7695
```

```
+---------------------+--------------------------------------+
| Field               | Value                                |
+---------------------+--------------------------------------+
| admin_state_up      | True                                 |
| binary              | neutron-linuxbridge-agent            |
| configurations      | {                                    |
|                     |      "physnet1": "eth0",             |
|                     |      "devices": "4"                  |
|                     | }                                    |
| created_at          | 2013-03-16T01:49:52.000000           |
| description         |                                      |
| disabled            | False                                |
| group               | agent                                |
| heartbeat_timestamp | 2013-03-16T01:59:45.000000           |
| host                | HostB                                |
| id                  | ed96b856-ae0f-4d75-bb28-40a47ffd7695 |
| topic               | N/A                                  |
| started_at          | 2013-03-16T06:48:39.000000           |
| type                | Linux bridge agent                   |
+---------------------+--------------------------------------+
```

The output shows `bridge-mapping` and the number of virtual network devices on this L2 agent.

Procedure 7.22. Manage assignment of networks to DHCP agent

Now that you have run the **net-list-on-dhcp-agent** and **dhcp-agent-list-hosting-net** commands, you can add a network to a DHCP agent and remove one from it.

1. Default scheduling.

   When you create a network with one port, you can schedule it to an active DHCP agent. If many active DHCP agents are running, select one randomly. You can design more sophisticated scheduling algorithms in the same way as `nova-schedule` later on.

   ```
   $ neutron net-create net2
   $ neutron subnet-create net2 9.0.1.0/24 --name subnet2
   $ neutron port-create net2
   $ neutron dhcp-agent-list-hosting-net net2
   +--------------------------------------+-------+----------------+-------+
   | id                                   | host  | admin_state_up | alive |
   +--------------------------------------+-------+----------------+-------+
   | a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True           | :-)   |
   +--------------------------------------+-------+----------------+-------+
   ```

   It is allocated to DHCP agent on HostA. If you want to validate the behavior through the **dnsmasq** command, you must create a subnet for the network because the DHCP agent starts the `dnsmasq` service only if there is a DHCP.

2. Assign a network to a given DHCP agent.

   To add another DHCP agent to host the network, run this command:

   ```
   $ neutron dhcp-agent-network-add f28aa126-6edb-4ea5-a81e-8850876bc0a8 net2
   Added network net2 to dhcp agent
   $ neutron dhcp-agent-list-hosting-net net2
   +--------------------------------------+-------+----------------+-------+
   ```

```
| id                                   | host  | admin_state_up | alive |
+--------------------------------------+-------+----------------+-------+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True           | :-)   |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | HostB | True           | :-)   |
+--------------------------------------+-------+----------------+-------+
```

Both DHCP agents host the `net2` network.

3. Remove a network from a specified DHCP agent.

   This command is the sibling command for the previous one. Remove `net2` from the DHCP agent for HostA:

```
$ neutron dhcp-agent-network-remove a0c1c21c-d4f4-4577-9ec7-908f2d48622d
 net2
Removed network net2 to dhcp agent
$ neutron dhcp-agent-list-hosting-net net2
+--------------------------------------+-------+----------------+-------+
| id                                   | host  | admin_state_up | alive |
+--------------------------------------+-------+----------------+-------+
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | HostB | True           | :-)   |
+--------------------------------------+-------+----------------+-------+
```

   You can see that only the DHCP agent for HostB is hosting the `net2` network.

Procedure 7.23. HA of DHCP agents

Boot a VM on net2. Let both DHCP agents host `net2`. Fail the agents in turn to see if the VM can still get the desired IP.

1. Boot a VM on net2.

```
$ neutron net-list
+--------------------------------------+------
+-----------------------------------------------------+
| id                                   | name | subnets
                       |
+--------------------------------------+------
+-----------------------------------------------------+
| 89dca1c6-c7d4-4f7a-b730-549af0fb6e34 | net1 | f6c832e3-9968-46fd-8e45-
d5cf646db9d1  10.0.1.0/24|
| 9b96b14f-71b8-4918-90aa-c5d705606b1a | net2 | 6979b71a-0ae8-448c-
aa87-65f68eedcaaa  9.0.1.0/24 |
+--------------------------------------+------
+-----------------------------------------------------+
$ nova boot --image tty --flavor 1 myserver4 \
  --nic net-id=9b96b14f-71b8-4918-90aa-c5d705606b1a
$ nova list
+--------------------------------------+-----------+--------
+---------------+
| ID                                   | Name      | Status | Networks
  |
+--------------------------------------+-----------+--------
+---------------+
| c394fcd0-0baa-43ae-a793-201815c3e8ce | myserver1 | ACTIVE | net1=10.0.1.
3 |
| 2d604e05-9a6c-4ddb-9082-8a1fbdcc797d | myserver2 | ACTIVE | net1=10.0.1.
4 |
| c7c0481c-3db8-4d7a-a948-60ce8211d585 | myserver3 | ACTIVE | net1=10.0.1.
5 |
| f62f4731-5591-46b1-9d74-f0c901de567f | myserver4 | ACTIVE | net2=9.0.1.2
  |
+--------------------------------------+-----------+--------
+---------------+
```

2. Make sure both DHCP agents hosting 'net2'.

   Use the previous commands to assign the network to agents.

```
$ neutron dhcp-agent-list-hosting-net net2
+--------------------------------------+-------+----------------+-------+
| id                                   | host  | admin_state_up | alive |
+--------------------------------------+-------+----------------+-------+
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | HostA | True           | :-)   |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | HostB | True           | :-)   |
+--------------------------------------+-------+----------------+-------+
```

Procedure 7.24. Test the HA

1. Log in to the `myserver4` VM, and run `udhcpc`, `dhclient` or other DHCP client.
2. Stop the DHCP agent on HostA. Besides stopping the `neutron-dhcp-agent` binary, you must stop the **dnsmasq** processes.
3. Run a DHCP client in VM to see if it can get the wanted IP.
4. Stop the DHCP agent on HostB too.
5. Run **udhcpc** in the VM; it cannot get the wanted IP.
6. Start DHCP agent on HostB. The VM gets the wanted IP again.

Procedure 7.25. Disable and remove an agent

An administrator might want to disable an agent if a system hardware or software upgrade is planned. Some agents that support scheduling also support disabling and enabling agents, such as L3 and DHCP agents. After the agent is disabled, the scheduler does not schedule new resources to the agent. After the agent is disabled, you can safely remove the agent. Remove the resources on the agent before you delete the agent.

- To run the following commands, you must stop the DHCP agent on HostA.

```
$ neutron agent-update --admin-state-up False a0c1c21c-
d4f4-4577-9ec7-908f2d48622d
$ neutron agent-list
+--------------------------------------+--------------------+-------
+-------+----------------+
| id                                   | agent_type         | host  |
 alive | admin_state_up |
+--------------------------------------+--------------------+-------
+-------+----------------+
| 1b69828d-6a9b-4826-87cd-1757f0e27f31 | Linux bridge agent | HostA | :-)
  | True           |
| a0c1c21c-d4f4-4577-9ec7-908f2d48622d | DHCP agent         | HostA | :-)
  | False          |
| ed96b856-ae0f-4d75-bb28-40a47ffd7695 | Linux bridge agent | HostB | :-)
  | True           |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent         | HostB | :-)
  | True           |
+--------------------------------------+--------------------+-------
+-------+----------------+
$ neutron agent-delete a0c1c21c-d4f4-4577-9ec7-908f2d48622d
Deleted agent: a0c1c21c-d4f4-4577-9ec7-908f2d48622d
$ neutron agent-list
+--------------------------------------+--------------------+-------
+-------+----------------+
| id                                   | agent_type         | host  |
 alive | admin_state_up |
+--------------------------------------+--------------------+-------
+-------+----------------+
| 1b69828d-6a9b-4826-87cd-1757f0e27f31 | Linux bridge agent | HostA | :-)
  | True           |
| ed96b856-ae0f-4d75-bb28-40a47ffd7695 | Linux bridge agent | HostB | :-)
  | True           |
| f28aa126-6edb-4ea5-a81e-8850876bc0a8 | DHCP agent         | HostB | :-)
  | True           |
+--------------------------------------+--------------------+-------
+-------+----------------+
```

After deletion, if you restart the DHCP agent, it appears on the agent list again.

Use Networking

You can manage OpenStack Networking services using the `service` command. For example:

```
# service neutron-server stop
# service neutron-server status
# service neutron-server start
# service neutron-server restart
```

Log files are in the `/var/log/neutron` directory.

Configuration files are in the `/etc/neutron` directory.

You can use Networking in the following ways:

- Expose the Networking API to cloud tenants, enabling them to build rich network topologies.
- Have the cloud administrator, or an automated administrative tool, create network connectivity on behalf of tenants.

A tenant or cloud administrator can both perform the following procedures.

Core Networking API features

After you install and run Networking, tenants and administrators can perform create-read-update-delete (CRUD) API networking operations by using the Networking API directly or the neutron command-line interface (CLI). The neutron CLI is a wrapper around the Networking API. Every Networking API call has a corresponding neutron command.

The CLI includes a number of options. For details, refer to the *OpenStack End User Guide*.

API abstractions

The Networking v2.0 API provides control over both L2 network topologies and their allocated IP addresses (IP Address Management or IPAM). There is also an extension to cover basic L3 forwarding and NAT, which provides capabilities similar to **nova-network**.

Table 7.15. API abstractions

Abstraction	Description
Network	An isolated L2 network segment (similar to a VLAN) that forms the basis for describing the L2 network topology available in an Networking deployment.
Subnet	Associates a block of IP addresses and other network configuration, such as default gateways or dns-servers, with an Networking network. Each subnet represents an IPv4 or IPv6 address block, and each Networking network can have multiple subnets.
Port	Represents an attachment port to a L2 Networking network. When a port is created on the network, by default it is allocated an available fixed IP address out of one of the designated subnets for each IP version (if one exists). When the port is destroyed, its allocated addresses return to the pool of available IPs on the subnet. Users of the Networking API can either choose a specific IP address from the block, or let Networking choose the first available IP address.

This table summarizes the attributes available for each networking abstraction. For information about API abstraction and operations, see the Networking API v2.0 Reference.

Table 7.16. Network attributes

Attribute	Type	Default value	Description
`admin_state_up`	bool	True	Administrative state of the network. If specified as False (down), this network does not forward packets.
`id`	uuid-str	Generated	UUID for this network.
`name`	string	None	Human-readable name for this network; is not required to be unique.
`shared`	bool	False	Specifies whether this network resource can be accessed by any tenant. The default policy setting restricts usage of this attribute to administrative users only.
`status`	string	N/A	Indicates whether this network is currently operational.
`subnets`	list(uuid-str)	Empty list	List of subnets associated with this network.
`tenant_id`	uuid-str	N/A	Tenant owner of the network. Only administrative users can set the tenant identifier; this cannot be changed using authorization policies.

Table 7.17. Subnet attributes

Attribute	Type	Default Value	Description
`allocation_pools`	list(dict)	Every address in `cidr`, excluding `gateway_ip` (if configured).	List of cidr sub-ranges that are available for dynamic allocation to ports. Syntax: `[ { "start":"10.0.0.2", "end": "10.0.0.254"} ]`
`cidr`	string	N/A	IP range for this subnet, based on the IP version.
`dns_nameservers`	list(string)	Empty list	List of DNS name servers used by hosts in this subnet.
`enable_dhcp`	bool	True	Specifies whether DHCP is enabled for this subnet.
`gateway_ip`	string	First address in `cidr`	Default gateway used by devices in this subnet.
`host_routes`	list(dict)	Empty list	Routes that should be used by devices with IPs from this subnet (not including local subnet route).
`id`	uuid-string	Generated	UUID representing this subnet.
`ip_version`	int	4	IP version.
`name`	string	None	Human-readable name for this subnet (might not be unique).
`network_id`	uuid-string	N/A	Network with which this subnet is associated.
`tenant_id`	uuid-string	N/A	Owner of network. Only administrative users can set the tenant identifier; this cannot be changed using authorization policies.

Table 7.18. Port attributes

Attribute	Type	Default Value	Description
`admin_state_up`	bool	true	Administrative state of this port. If specified as False (down), this port does not forward packets.
`device_id`	string	None	Identifies the device using this port (for example, a virtual server's ID).
`device_owner`	string	None	Identifies the entity using this port (for example, a dhcp agent).
`fixed_ips`	list(dict)	Automatically allocated from pool	Specifies IP addresses for this port; associates the port with the subnets containing the listed IP addresses.
`id`	uuid-string	Generated	UUID for this port.
`mac_address`	string	Generated	Mac address to use on this port.
`name`	string	None	Human-readable name for this port (might not be unique).
`network_id`	uuid-string	N/A	Network with which this port is associated.
`status`	string	N/A	Indicates whether the network is currently operational.
`tenant_id`	uuid-string	N/A	Owner of the network. Only administrative users can set the tenant identifier; this cannot be changed using authorization policies.

Basic Networking operations

To learn about advanced capabilities available through the neutron command-line interface (CLI), read the networking section in the OpenStack End User Guide.

This table shows example neutron commands that enable you to complete basic network operations:

Table 7.19. Basic Networking operations

Operation	Command
Creates a network.	`$ neutron net-create net1`
Creates a subnet that is associated with net1.	`$ neutron subnet-create net1 10.0.0.0/24`
Lists ports for a specified tenant.	`$ neutron port-list`
Lists ports for a specified tenant and displays the `id`, `fixed_ips`, and `device_owner` columns.	`$ neutron port-list -c id -c fixed_ips -c device_owner`
Shows information for a specified port.	`$ neutron port-show *port-id*`

Note

The `device_owner` field describes who owns the port. A port whose `device_owner` begins with:

- `network` is created by Networking.
- `compute` is created by Compute.

Administrative operations

The cloud administrator can run any **neutron** command on behalf of tenants by specifying an Identity `tenant_id` in the command, as follows:

```
$ neutron net-create --tenant-id=tenant-id network-name
```

For example:

```
$ neutron net-create --tenant-id=5e4bbe24b67a4410bc4d9fae29ec394e net1
```

Note

To view all tenant IDs in Identity, run the following command as an Identity Service admin user:

```
$ keystone tenant-list
```

Advanced Networking operations

This table shows example Networking commands that enable you to complete advanced network operations:

Table 7.20. Advanced Networking operations

Operation	Command
Creates a network that all tenants can use.	`$ neutron net-create --shared public-net`
Creates a subnet with a specified gateway IP address.	`$ neutron subnet-create --gateway 10.0.0.254 net1 10.0.0.0/24`
Creates a subnet that has no gateway IP address.	`$ neutron subnet-create --no-gateway net1 10.0.0.0/24`
Creates a subnet with DHCP disabled.	`$ neutron subnet-create net1 10.0.0.0/24 --enable_dhcp False`
Creates a subnet with a specified set of host routes.	`$ neutron subnet-create test-net1 40.0.0.0/24 --host_routes type=dict list=true destination=40.0.1.0/24,nexthop=40.0.0.2`
Creates a subnet with a specified set of dns name servers.	`$ neutron subnet-create test-net1 40.0.0.0/24 --dns_nameservers list=true 8.8.8.7 8.8.8.8`
Displays all ports and IPs allocated on a network.	`$ neutron port-list --network_id` ***`net-id`***

Use Compute with Networking

Basic Compute and Networking operations

This table shows example neutron and nova commands that enable you to complete basic VM networking operations:

Table 7.21. Basic Compute and Networking operations

Action	Command
Checks available networks.	`$ neutron net-list`
Boots a VM with a single NIC on a selected Networking network.	`$ nova boot --image img --flavor flavor --nic net-id=net-id vm-name`
Searches for ports with a `device_id` that matches the Compute instance UUID. See Create and delete VMs [234].	`$ neutron port-list --device_id=vm-id`
Searches for ports, but shows only the `mac_address` of the port.	`$ neutron port-list --field mac_address --device_id=vm-id`
Temporarily disables a port from sending traffic.	`$ neutron port-update port-id --admin_state_up=False`

Note

The `device_id` can also be a logical router ID.

Create and delete VMs

- When you boot a Compute VM, a port on the network that corresponds to the VM NIC is automatically created and associated with the default security group. You can configure security group rules to enable users to access the VM.
- When you delete a Compute VM, the underlying Networking port is automatically deleted.

Advanced VM creation operations

This table shows example nova and neutron commands that enable you to complete advanced VM creation operations:

Table 7.22. Advanced VM creation operations

Operation	Command
Boots a VM with multiple NICs.	`$ nova boot --image img --flavor flavor --nic net-id=net1-id --nic net-id=net2-id vm-name`

Boots a VM with a specific IP address. First, create an Networking port with a specific IP address. Then, boot a VM specifying a `port-id` rather than a `net-id`.	`$ neutron port-create --fixed-ip subnet_id=subnet-id, ip_address=IP net-id`
	`$ nova boot --image img --flavor flavor --nic port-id=port-id vm-name`

Boots a VM that connects to all networks that are accessible to the tenant who submits the request (without the `--nic` option).	`$ nova boot --image img --flavor flavor vm-name`

Note

Networking does not currently support the *`v4-fixed-ip`* parameter of the `--nic` option for the **nova** command.

Enable ping and SSH on VMs (security groups)

You must configure security group rules depending on the type of plug-in you are using. If you are using a plug-in that:

- Implements Networking security groups, you can configure security group rules directly by using **neutron security-group-rule-create**. This example enables **ping** and **ssh** access to your VMs.

```
$ neutron security-group-rule-create --protocol icmp \
    --direction ingress default
```

```
$ neutron security-group-rule-create --protocol tcp --port-range-min 22 \
    --port-range-max 22 --direction ingress default
```

- Does not implement Networking security groups, you can configure security group rules by using the **nova secgroup-add-rule** or **euca-authorize** command. These **nova** commands enable **ping** and **ssh** access to your VMs.

```
$ nova secgroup-add-rule default icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 0.0.0.0/0
```

Note

If your plug-in implements Networking security groups, you can also leverage Compute security groups by setting `security_group_api = neutron` in the `nova.conf` file. After you set this option, all Compute security group commands are proxied to Networking.

Advanced features through API extensions

Several plug-ins implement API extensions that provide capabilities similar to what was available in nova-network: These plug-ins are likely to be of interest to the OpenStack community.

Provider networks

Networks can be categorized as either "tenant networks" or "provider networks". Tenant networks are created by normal users, and details about how they are physically realized are hidden from those users. Provider networks are created with administrative credentials, specifying the details of how the network is physically realized, usually to match some existing network in the data center.

Provider networks enable cloud administrators to create Networking networks that map directly to the physical networks in the data center. This is commonly used to give tenants direct access to a public network that can be used to reach the Internet. It might also be used to integrate with VLANs in the network that already have a defined meaning (for example, enable a VM from the "marketing" department to be placed on the same VLAN as bare-metal marketing hosts in the same data center).

The provider extension allows administrators to explicitly manage the relationship between Networking virtual networks and underlying physical mechanisms such as VLANs and tunnels. When this extension is supported, Networking client users with administrative privileges see additional provider attributes on all virtual networks, and are able to specify these attributes in order to create provider networks.

The provider extension is supported by the Open vSwitch and Linux Bridge plug-ins. Configuration of these plug-ins requires familiarity with this extension.

Terminology

A number of terms are used in the provider extension and in the configuration of plug-ins supporting the provider extension:

Table 7.23. Provider extension terminology

Term	Description
virtual network	An Networking L2 network (identified by a UUID and optional name) whose ports can be attached as vNICs to Compute instances and to various Networking agents. The Open vSwitch and Linux Bridge plug-ins each support several different mechanisms to realize virtual networks.
physical network	A network connecting virtualization hosts (such as compute nodes) with each other and with other network resources. Each physical network might support multiple virtual networks. The provider extension and the plug-in configurations identify physical networks using simple string names.
tenant network	A virtual network that a tenant or an administrator creates. The physical details of the network are not exposed to the tenant.
provider network	A virtual network administratively created to map to a specific network in the data center, typically to enable direct access to non-OpenStack resources on that network. Tenants can be given access to provider networks.
VLAN network	A virtual network implemented as packets on a specific physical network containing IEEE 802.1Q headers with a specific VID field value. VLAN networks sharing the same physical network are isolated from each other at L2, and can even have overlapping IP address spaces. Each distinct physical network supporting VLAN networks is treated as a separate VLAN trunk, with a distinct space of VID values. Valid VID values are 1 through 4094.
flat network	A virtual network implemented as packets on a specific physical network containing no IEEE 802.1Q header. Each physical network can realize at most one flat network.
local network	A virtual network that allows communication within each host, but not across a network. Local networks are intended mainly for single-node test scenarios, but can have other uses.
GRE network	A virtual network implemented as network packets encapsulated using GRE. GRE networks are also referred to as *tunnels*. GRE tunnel packets are routed by the IP routing table for the host, so GRE networks are not associated by Networking with specific physical networks.
Virtual Extensible LAN (VXLAN) network	VXLAN is a proposed encapsulation protocol for running an overlay network on existing Layer 3 infrastructure. An overlay network is a virtual network that is built on top of existing network Layer 2 and Layer 3 technologies to support elastic compute architectures.

The ML2, Open vSwitch, and Linux Bridge plug-ins support VLAN networks, flat networks, and local networks. Only the ML2 and Open vSwitch plug-ins currently support GRE and VXLAN networks, provided that the required features exist in the hosts Linux kernel, Open vSwitch, and iproute2 packages.

Provider attributes

The provider extension extends the Networking network resource with these attributes:

Table 7.24. Provider network attributes

Attribute name	Type	Default Value	Description
provider:network_type	String	N/A	The physical mechanism by which the virtual network is implemented. Possible values are `flat`, `vlan`, `local`, and `gre`, corresponding to flat networks, VLAN networks, local networks, and GRE networks as defined above. All types of provider networks can be created by administrators, while tenant networks can be implemented as `vlan`, `gre`, or `local` network types depending on plug-in configuration.
provider:physical_network	String	If a physical network named "default" has been configured, and if provider:network_type is `flat` or `vlan`, then "default" is used.	The name of the physical network over which the virtual network is implemented for flat and VLAN networks. Not applicable to the `local` or `gre` network types.
provider:segmentation_id	Integer	N/A	For VLAN networks, the VLAN VID on the physical network that realizes the virtual network. Valid VLAN VIDs are 1 through 4094. For GRE networks, the tunnel ID. Valid tunnel IDs are any 32 bit unsigned integer. Not applicable to the `flat` or `local` network types.

To view or set provider extended attributes, a client must be authorized for the `extension:provider_network:view` and `extension:provider_network:set` actions in the Networking policy configuration. The default Networking configuration authorizes both actions for users with the admin role. An authorized client or an administrative user can view and set the provider extended attributes through Networking API calls. See the section called "Authentication and authorization" [255] for details on policy configuration.

Provider extension API operations

To use the provider extension with the default policy settings, you must have the administrative role.

This list shows example neutron commands that enable you to complete basic provider extension API operations:

- Shows all attributes of a network, including provider attributes:

```
$ neutron net-show <name or net-id>
```

- Creates a local provider network:

```
$ neutron net-create <name> --tenant_id <tenant-id> --provider:network_type
 local
```

- When you create flat networks, <phys-net-name> must be known to the plug-in. See the *OpenStack Configuration Reference* for details. Creates a flat provider network:

```
$ neutron net-create <name> --tenant_id <tenant-id> --provider:network_type
 flat --provider:physical_network <phys-net-name>
```

- When you create VLAN networks, <phys-net-name> must be known to the plug-in. See the *OpenStack Configuration Reference* for details on configuring network_vlan_ranges to identify all physical networks. When you create VLAN networks, <VID> can fall either within or outside any configured ranges of VLAN IDs from which tenant networks are allocated. Creates a VLAN provider network:

```
$ neutron net-create <name> --tenant_id <tenant-id> --provider:network_type
 vlan --provider:physical_network <phys-net-name> --provider:segmentation_id
 <VID>
```

- When you create GRE networks, <tunnel-id> can be either inside or outside any tunnel ID ranges from which tenant networks are allocated.

  After you create provider networks, you can allocate subnets, which you can use in the same way as other virtual networks, subject to authorization policy based on the specified <tenant_id>. Creates a GRE provider network:

  ```
  $ neutron net-create <name> --tenant_id <tenant-id> --provider:network_type
   gre --provider:segmentation_id <tunnel-id>
  ```

L3 routing and NAT

The Networking API provides abstract L2 network segments that are decoupled from the technology used to implement the L2 network. Networking includes an API extension that provides abstract L3 routers that API users can dynamically provision and configure. These Networking routers can connect multiple L2 Networking networks, and can also provide a gateway that connects one or more private L2 networks to a shared external network. For example, a public network for access to the Internet. See the *OpenStack Configuration Reference* for details on common models of deploying Networking L3 routers.

The L3 router provides basic NAT capabilities on gateway ports that uplink the router to external networks. This router SNATs all traffic by default, and supports floating IPs, which creates a static one-to-one mapping from a public IP on the external network to a private IP on one of the other subnets attached to the router. This allows a tenant to selectively expose VMs on private networks to other hosts on the external network (and often to all hosts on the Internet). You can allocate and map floating IPs from one port to another, as needed.

L3 API abstractions

Table 7.25. Router

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the router.
name	String	None	Human-readable name for the router. Might not be unique.
admin_state_up	Bool	True	The administrative state of router. If false (down), the router does not forward packets.
status	String	N/A	Indicates whether router is currently operational.
tenant_id	uuid-str	N/A	Owner of the router. Only admin users can specify a tenant_id other than its own.
external_gateway_info	dict contain 'network_id' key-value pair	Null	External network that this router connects to for gateway services (for example, NAT)

Table 7.26. Floating IP

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the floating IP.
floating_ip_address	string (IP address)	allocated by Networking	The external network IP address available to be mapped to an internal IP address.
floating_network_id	uuid-str	N/A	The network indicating the set of subnets from which the floating IP should be allocated
router_id	uuid-str	N/A	Read-only value indicating the router that connects the external network to the associated internal port, if a port is associated.
port_id	uuid-str	Null	Indicates the internal Networking port associated with the external floating IP.
fixed_ip_address	string (IP address)	Null	Indicates the IP address on the internal port that is mapped to by the floating IP (since an Networking port might have more than one IP address).
tenant_id	uuid-str	N/A	Owner of the Floating IP. Only admin users can specify a tenant_id other than its own.

Basic L3 operations

External networks are visible to all users. However, the default policy settings enable only administrative users to create, update, and delete external networks.

This table shows example neutron commands that enable you to complete basic L3 operations:

Table 7.27. Basic L3 operations

Operation	Command

Creates external networks.	`# neutron net-create public --router:external=True`
	`$ neutron subnet-create public 172.16.1.0/24`

Lists external networks.	`$ neutron net-list -- --router:external=True`

Creates an internal-only router that connects to multiple L2 networks privately.	`$ neutron net-create net1`
	`$ neutron subnet-create net1 10.0.0.0/24`
	`$ neutron net-create net2`
	`$ neutron subnet-create net2 10.0.1.0/24`
	`$ neutron router-create router1`
	`$ neutron router-interface-add router1 <subnet1-uuid>`
	`$ neutron router-interface-add router1 <subnet2-uuid>`

Connects a router to an external network, which enables that router to act as a NAT gateway for external connectivity.	`$ neutron router-gateway-set router1 <ext-net-id>` The router obtains an interface with the gateway_ip address of the subnet, and this interface is attached to a port on the L2 Networking network associated with the subnet. The router also gets a gateway interface to the specified external network. This provides SNAT connectivity to the external network as well as support for floating IPs allocated on that external networks. Commonly an external network maps to a network in the provider
Lists routers.	`$ neutron router-list`
Shows information for a specified router.	`$ neutron router-show <router_id>`
Shows all internal interfaces for a router.	

Identifies the `port-id` that represents the VM NIC to which the floating IP should map.	`$ neutron port-list -c id -c fixed_ips -- --device_id=`
	`<instance_id>`
	
	This port must be on an Networking subnet that is attached to a router uplinked to the external network used to create the floating IP. Conceptually, this is because the router must be able to perform the Destination NAT (DNAT) rewriting of packets from the Floating IP address (chosen from a subnet on the external network) to the internal Fixed IP (chosen from a private subnet that is behind the router).

Creates a floating IP address and associates it with a port.	`$ neutron floatingip-create <ext-net-id>`
	`$ neutron floatingip-associate <floatingip-id> <internal VM`
	` port-id>`

Creates a floating IP address and associates it with a port, in a single step.	`$ neutron floatingip-create --port_id <internal VM port-id>`
	` <ext-net-id>`

Lists floating IPs.	`$ neutron floatingip-list`
Finds floating IP for a specified VM port.	`$ neutron floatingip-list -- --port_id=ZZZ`
Disassociates a floating IP address.	`$ neutron floatingip-disassociate <floatingip-id>`
Deletes the floating IP address.	`$ neutron floatingip-delete <floatingip-id>`
Clears the gateway.	`$ neutron router-gateway-clear router1`
Removes the interfaces from the router.	`$ neutron router-interface-delete router1 <subnet-id>`
Deletes the router.	`$ neutron router-delete router1`

Security groups

Security groups and security group rules allows administrators and tenants the ability to specify the type of traffic and direction (ingress/egress) that is allowed to pass through a port. A security group is a container for security group rules.

When a port is created in Networking it is associated with a security group. If a security group is not specified the port is associated with a 'default' security group. By default, this group drops all ingress traffic and allows all egress. Rules can be added to this group in order to change the behaviour.

To use the Compute security group APIs or use Compute to orchestrate the creation of ports for instances on specific security groups, you must complete additional configuration. You must configure the `/etc/nova/nova.conf` file and set the `security_group_api=neutron` option on every node that runs `nova-compute` and `nova-api`. After you make this change, restart `nova-api` and `nova-compute` to pick up this change. Then, you can use both the Compute and OpenStack Network security group APIs at the same time.

Note

- To use the Compute security group API with Networking, the Networking plug-in must implement the security group API. The following plug-ins currently implement this: ML2, Open vSwitch, Linux Bridge, NEC, Ryu, and VMware NSX.
- You must configure the correct firewall driver in the `securitygroup` section of the plug-in/agent configuration file. Some plug-ins and agents, such as Linux Bridge Agent and Open vSwitch Agent, use the no-operation driver as the default, which results in non-working security groups.
- When using the security group API through Compute, security groups are applied to all ports on an instance. The reason for this is that Compute security group APIs are instances based and not port based as Networking.

Security group API abstractions

Table 7.28. Security group attributes

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the security group.
name	String	None	Human-readable name for the security group. Might not be unique. Cannot be named default as that is automatically created for a tenant.
description	String	None	Human-readable description of a security group.
tenant_id	uuid-str	N/A	Owner of the security group. Only admin users can specify a tenant_id other than their own.

Table 7.29. Security group rules

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the security group rule.

Attribute name	Type	Default Value	Description
security_group_id	uuid-str or Integer	allocated by Networking	The security group to associate rule with.
direction	String	N/A	The direction the traffic is allow (ingress/ egress) from a VM.
protocol	String	None	IP Protocol (icmp, tcp, udp, and so on).
port_range_min	Integer	None	Port at start of range
port_range_max	Integer	None	Port at end of range
ethertype	String	None	ethertype in L2 packet (IPv4, IPv6, and so on)
remote_ip_prefix	string (IP cidr)	None	CIDR for address range
remote_group_id	uuid-str or Integer	allocated by Networking or Compute	Source security group to apply to rule.
tenant_id	uuid-str	N/A	Owner of the security group rule. Only admin users can specify a tenant_id other than its own.

Basic security group operations

This table shows example neutron commands that enable you to complete basic security group operations:

Table 7.30. Basic security group operations

Operation	Command
Creates a security group for our web servers.	`$ neutron security-group-create webservers --description "security group for webservers"`
Lists security groups.	`$ neutron security-group-list`
Creates a security group rule to allow port 80 ingress.	`$ neutron security-group-rule-create --direction ingress --protocol tcp --port_range_min 80 --port_range_max 80 <security_group_uuid>`
Lists security group rules.	`$ neutron security-group-rule-list`
Deletes a security group rule.	`$ neutron security-group-rule-delete <security_group_rule_uuid>`
Deletes a security group.	`$ neutron security-group-delete <security_group_uuid>`
Creates a port and associates two security groups.	`$ neutron port-create --security-group <security_group_id1> --security-group <security_group_id2> <network_id>`
Removes security groups from a port.	`$ neutron port-update --no-security-groups <port_id>`

Basic Load-Balancer-as-a-Service operations

Note

The Load-Balancer-as-a-Service (LBaaS) API provisions and configures load balancers. The Havana release offers a reference implementation that is based on the HAProxy software load balancer.

This list shows example neutron commands that enable you to complete basic LBaaS operations:

- Creates a load balancer pool by using specific provider.

  `--provider` is an optional argument. If not used, the pool is created with default provider for LBaaS service. You should configure the default provider in the `[service_providers]` section of `neutron.conf` file. If no default provider is specified for LBaaS, the `--provider` option is required for pool creation.

  ```
  $ neutron lb-pool-create --lb-method ROUND_ROBIN --name mypool --protocol
   HTTP --subnet-id <subnet-uuid> --provider <provider_name>
  ```

- Associates two web servers with pool.

  ```
  $ neutron lb-member-create --address  <webserver one IP> --protocol-port 80
   mypool
  $ neutron lb-member-create --address  <webserver two IP> --protocol-port 80
   mypool
  ```

- Creates a health monitor which checks to make sure our instances are still running on the specified protocol-port.

  ```
  $ neutron lb-healthmonitor-create --delay 3 --type HTTP --max-retries 3 --
  timeout 3
  ```

- Associates a health monitor with pool.

  ```
  $ neutron lb-healthmonitor-associate  <healthmonitor-uuid> mypool
  ```

- Creates a virtual IP (VIP) address that, when accessed through the load balancer, directs the requests to one of the pool members.

  ```
  $ neutron lb-vip-create --name myvip --protocol-port 80 --protocol HTTP --
  subnet-id <subnet-uuid> mypool
  ```

Firewall-as-a-Service

The Firewall-as-a-Service (FWaaS) API is an experimental API that enables early adopters and vendors to test their networking implementations.

Firewall-as-a-Service API abstractions

Table 7.31. Firewall rules

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the firewall rule.
tenant_id	uuid-str	N/A	Owner of the firewall rule. Only admin users can specify a tenant_id other than its own.
name	String	None	Human readable name for the firewall rule (255 characters limit).
description	String	None	Human readable description for the firewall rule (1024 characters limit).
firewall_policy_id	uuid-str or None	allocated by Networking	This is a read-only attribute that gets populated with the uuid of the firewall policy when this firewall rule is associated with a firewall policy. A firewall rule can be associated with only one firewall policy at a time. However, the association can be changed to a different firewall policy.
shared	Boolean	False	When set to True makes this firewall rule visible to tenants other than its owner, and it can be used in firewall policies not owned by its tenant.
protocol	String	None	IP Protocol (icmp, tcp, udp, None).
ip_version	Integer or String	4	IP Version (4, 6).
source_ip_address	String (IP address or CIDR)	None	Source IP address or CIDR.
destination_ip_address	String (IP address or CIDR)	None	Destination IP address or CIDR.
source_port	Integer or String (either as a single port number or in the format of a ':' separated range)	None	Source port number or a range.
destination_port	Integer or String (either as a single port number or in the format of a ':' separated range)	None	Destination port number or a range.
position	Integer	None	This is a read-only attribute that gets assigned to this rule when the rule is associated with a firewall policy. It indicates the position of this rule in that firewall policy.
action	String	deny	Action to be performed on the traffic matching the rule (allow, deny).
enabled	Boolean	True	When set to False, disables this rule in the firewall policy. Facilitates selectively turning off rules without having to disassociate the rule from the firewall policy.

Table 7.32. Firewall policies

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the firewall policy.
tenant_id	uuid-str	N/A	Owner of the firewall policy. Only admin users can specify a tenant_id other their own.
name	String	None	Human readable name for the firewall policy (255 characters limit).
description	String	None	Human readable description for the firewall policy (1024 characters limit).
shared	Boolean	False	When set to True makes this firewall policy visible to tenants other than its owner, and can be used to associate with firewalls not owned by its tenant.
firewall_rules	List of uuid-str or None	None	This is an ordered list of firewall rule uuids. The firewall applies the rules in the order in which they appear in this list.
audited	Boolean	False	When set to True by the policy owner indicates that the firewall policy has been audited. This attribute is meant to aid in the firewall policy audit workflows. Each time the firewall policy or the associated firewall rules are changed, this attribute is set to False and must be explicitly set to True through an update operation.

Table 7.33. Firewalls

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the firewall.
tenant_id	uuid-str	N/A	Owner of the firewall. Only admin users can specify a tenant_id other than its own.
name	String	None	Human readable name for the firewall (255 characters limit).
description	String	None	Human readable description for the firewall (1024 characters limit).
admin_state_up	Boolean	True	The administrative state of the firewall. If False (down), the firewall does not forward any packets.
status	String	N/A	Indicates whether the firewall is currently operational. Possible values include: • ACTIVE • DOWN • PENDING_CREATE • PENDING_UPDATE • PENDING_DELETE • ERROR
firewall_policy_id	uuid-str or None	None	The firewall policy uuid that this firewall is associated with. This firewall implements the rules contained in the firewall policy represented by this uuid.

Plug-in specific extensions

Each vendor can choose to implement additional API extensions to the core API. This section describes the extensions for each plug-in.

VMware NSX extensions

These sections explain NSX plug-in extensions.

VMware NSX QoS extension

The VMware NSX QoS extension rate-limits network ports to guarantee a specific amount of bandwidth for each port. This extension, by default, is only accessible by a tenant with an admin role but is configurable through the `policy.json` file. To use this extension, create a queue and specify the min/max bandwidth rates (kbps) and optionally set the QoS Marking and DSCP value (if your network fabric uses these values to make forwarding decisions). Once created, you can associate a queue with a network. Then, when ports are created on that network they are automatically created and associated with the specific queue size that was associated with the network. Because one size queue for a every port on a network might not be optimal, a scaling factor from the Nova flavor 'rxtx_factor' is passed in from Compute when creating the port to scale the queue.

Lastly, if you want to set a specific baseline QoS policy for the amount of bandwidth a single port can use (unless a network queue is specified with the network a port is created on) a default queue can be created in Networking which then causes ports created to be associated with a queue of that size times the rxtx scaling factor. Note that after a network or default queue is specified, queues are added to ports that are subsequently created but are not added to existing ports.

VMware NSX QoS API abstractions

Table 7.34. VMware NSX QoS attributes

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the QoS queue.
default	Boolean	False by default	If True, ports are created with this queue size unless the network port is created or associated with a queue at port creation time.
name	String	None	Name for QoS queue.
min	Integer	0	Minimum Bandwidth Rate (kbps).
max	Integer	N/A	Maximum Bandwidth Rate (kbps).
qos_marking	String	untrusted by default	Whether QoS marking should be trusted or untrusted.
dscp	Integer	0	DSCP Marking value.
tenant_id	uuid-str	N/A	The owner of the QoS queue.

Basic VMware NSX QoS operations

This table shows example neutron commands that enable you to complete basic queue operations:

Table 7.35. Basic VMware NSX QoS operations

Operation	Command
Creates QoS Queue (admin-only).	`$ neutron queue-create--min 10 --max 1000 myqueue`
Associates a queue with a network.	`$ neutron net-create network --queue_id=<queue_id>`
Creates a default system queue.	`$ neutron queue-create --default True --min 10 --max 2000 default`
Lists QoS queues.	`$ neutron queue-list`
Deletes a QoS queue.	`$ neutron queue-delete <queue_id or name>'`

VMware NSX provider networks extension

Provider networks can be implemented in different ways by the underlying NSX platform.

The *FLAT* and *VLAN* network types use bridged transport connectors. These network types enable the attachment of large number of ports. To handle the increased scale, the NSX plug-in can back a single OpenStack Network with a chain of NSX logical switches. You can specify the maximum number of ports on each logical switch in this chain on the `max_lp_per_bridged_ls` parameter, which has a default value of 5,000.

The recommended value for this parameter varies with the NSX version running in the back-end, as shown in the following table.

Table 7.36. Recommended values for max_lp_per_bridged_ls

NSX version	Recommended Value
2.x	64
3.0.x	5,000
3.1.x	5,000
3.2.x	10,000

In addition to these network types, the NSX plug-in also supports a special *l3_ext* network type, which maps external networks to specific NSX gateway services as discussed in the next section.

VMware NSX L3 extension

NSX exposes its L3 capabilities through gateway services which are usually configured out of band from OpenStack. To use NSX with L3 capabilities, first create a L3 gateway service in the NSX Manager. Next, in `/etc/neutron/plugins/vmware/nsx.ini` set `default_l3_gw_service_uuid` to this value. By default, routers are mapped to this gateway service.

VMware NSX L3 extension operations

Create external network and map it to a specific NSX gateway service:

```
$ neutron net-create public --router:external=True --provider:network_type
 l3_ext \
--provider:physical_network <L3-Gateway-Service-UUID>
```

Terminate traffic on a specific VLAN from a NSX gateway service:

```
$ neutron net-create public --router:external=True --provider:network_type
 l3_ext \
--provider:physical_network <L3-Gateway-Service-UUID> --
provider:segmentation_id <VLAN_ID>
```

Operational status synchronization in the VMware NSX plug-in

Starting with the Havana release, the VMware NSX plug-in provides an asynchronous mechanism for retrieving the operational status for neutron resources from the NSX back-end; this applies to *network*, *port*, and *router* resources.

The back-end is polled periodically, and the status for every resource is retrieved; then the status in the Networking database is updated only for the resources for which a status change occurred. As operational status is now retrieved asynchronously, performance for GET operations is consistently improved.

Data to retrieve from the back-end are divided in chunks in order to avoid expensive API requests; this is achieved leveraging NSX APIs response paging capabilities. The minimum chunk size can be specified using a configuration option; the actual chunk size is then determined dynamically according to: total number of resources to retrieve, interval between two synchronization task runs, minimum delay between two subsequent requests to the NSX back-end.

The operational status synchronization can be tuned or disabled using the configuration options reported in this table; it is however worth noting that the default values work fine in most cases.

Table 7.37. Configuration options for tuning operational status synchronization in the NSX plug-in

Option name	Group	Default value	Type and constraints	Notes
`state_sync_interval`	`nsx_sync`	120 seconds	Integer; no constraint.	Interval in seconds between two run of the synchronization task. If the synchronization task takes more than `state_sync_interval` seconds to execute, a new instance of the task is started as soon as the other is completed. Setting the value for this option to 0 will disable the synchronization task.
`max_random_sync_delay`	`nsx_sync`	0 seconds	Integer. Must not exceed `min_sync_req_delay`	When different from zero, a random delay between 0 and `max_random_sync_delay` will be added before processing the next chunk.
`min_sync_req_delay`	`nsx_sync`	10 seconds	Integer. Must not exceed `state_sync_interval`.	The value of this option can be tuned according to the observed load on the NSX controllers. Lower values will result in faster synchronization, but might increase the load on the controller cluster.
`min_chunk_size`	`nsx_sync`	500 resources	Integer; no constraint.	Minimum number of resources to retrieve from the back-end for each synchronization chunk. The expected number of synchronization chunks is given by the ratio between `state_sync_interval` and `min_sync_req_delay`. This size of a chunk might increase if the total number of resources is such that more than `min_chunk_size` resources must be fetched in one chunk with the current number of chunks.
`always_read_status`	`nsx_sync`	False	Boolean; no constraint.	When this option is enabled, the operational status will always be retrieved from the NSX back-end ad every GET request. In this case it is advisable to disable the synchronization task.

When running multiple OpenStack Networking server instances, the status synchronization task should not run on every node; doing so sends unnecessary traffic to the NSX back-end and performs unnecessary DB operations. Set the `state_sync_interval` configuration option to a non-zero value exclusively on a node designated for back-end status synchronization.

The *`fields=status`* parameter in Networking API requests always triggers an explicit query to the NSX back end, even when you enable asynchronous state synchronization. For example, `GET /v2.0/networks/<net-id>?fields=status&fields=name`.

Big Switch plug-in extensions

This section explains the Big Switch Neutron plug-in-specific extension.

Big Switch router rules

Big Switch allows router rules to be added to each tenant router. These rules can be used to enforce routing policies such as denying traffic between subnets or traffic to external networks. By enforcing these at the router level, network segmentation policies can be enforced across many VMs that have differing security groups.

Router rule attributes

Each tenant router has a set of router rules associated with it. Each router rule has the attributes in this table. Router rules and their attributes can be set using the **neutron router-update** command, through the Horizon interface or the Neutron API.

Table 7.38. Big Switch Router rule attributes

Attribute name	Required	Input Type	Description
source	Yes	A valid CIDR or one of the keywords 'any' or 'external'	The network that a packet's source IP must match for the rule to be applied
destination	Yes	A valid CIDR or one of the keywords 'any' or 'external'	The network that a packet's destination IP must match for the rule to be applied
action	Yes	'permit' or 'deny'	Determines whether or not the matched packets will allowed to cross the router
nexthop	No	A plus-separated (+) list of next-hop IP addresses. For example, `1.1.1.1+1.1.1.2`.	Overrides the default virtual router used to handle traffic for packets that match the rule

Order of rule processing

The order of router rules has no effect. Overlapping rules are evaluated using longest prefix matching on the source and destination fields. The source field is matched first so it always takes higher precedence over the destination field. In other words, longest prefix matching is used on the destination field only if there are multiple matching rules with the same source.

Big Switch router rules operations

Router rules are configured with a router update operation in OpenStack Networking. The update overrides any previous rules so all rules must be provided at the same time.

Update a router with rules to permit traffic by default but block traffic from external networks to the 10.10.10.0/24 subnet:

```
$ neutron router-update Router-UUID --router_rules type=dict list=true\
source=any,destination=any,action=permit \
source=external,destination=10.10.10.0/24,action=deny
```

Specify alternate next-hop addresses for a specific subnet:

```
$ neutron router-update Router-UUID --router_rules type=dict list=true\
source=any,destination=any,action=permit \
source=10.10.10.0/24,destination=any,action=permit,nexthops=10.10.10.254+10.
10.10.253
```

Block traffic between two subnets while allowing everything else:

```
$ neutron router-update Router-UUID --router_rules type=dict list=true\
source=any,destination=any,action=permit \
source=10.10.10.0/24,destination=10.20.20.20/24,action=deny
```

L3 metering

The L3 metering API extension enables administrators to configure IP ranges and assign a specified label to them to be able to measure traffic that goes through a virtual router.

The L3 metering extension is decoupled from the technology that implements the measurement. Two abstractions have been added: One is the metering label that can contain metering rules. Because a metering label is associated with a tenant, all virtual routers in this tenant are associated with this label.

L3 metering API abstractions

Table 7.39. Label

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the metering label.
name	String	None	Human-readable name for the metering label. Might not be unique.
description	String	None	The optional description for the metering label.
tenant_id	uuid-str	N/A	Owner of the metering label.

Table 7.40. Rules

Attribute name	Type	Default Value	Description
id	uuid-str	generated	UUID for the metering rule.
direction	String (Either ingress or egress)	ingress	The direction in which metering rule is applied, either ingress or egress.
metering_label_id	uuid-str	N/A	The metering label ID to associate with this metering rule.
excluded	Boolean	False	Specify whether the remote_ip_prefix will be excluded or not from traffic counters of the metering label (for example, to not count the traffic of a specific IP address of a range).

Attribute name	Type	Default Value	Description
remote_ip_prefix	String (CIDR)	N/A	Indicates remote IP prefix to be associated with this metering rule.

Basic L3 metering operations

Only administrators can manage the L3 metering labels and rules.

This table shows example **neutron** commands that enable you to complete basic L3 metering operations:

Table 7.41. Basic L3 operations

Operation	Command
Creates a metering label.	`$ neutron meter-label-create label1 --description "description of label1"`
Lists metering labels.	`$ neutron meter-label-list`

Shows information for a specified label.	`$ neutron meter-label-show label-uuid`
	`$ neutron meter-label-show label1`

Deletes a metering label.	`$ neutron meter-label-delete label-uuid`
	`$ neutron meter-label-delete label1`

Creates a metering rule.	`$ neutron meter-label-rule-create label-uuid cidr --direction direction --excluded`
	`$ neutron meter-label-rule-create label1 10.0.0.0/24 --direction ingress`
	`$ neutron meter-label-rule-create label1 20.0.0.0/24 --excluded`

Lists metering all label rules.	`$ neutron meter-label-rule-list`
Shows information for a specified label rule.	`$ neutron meter-label-rule-show rule-uuid`
Deletes a metering label rule.	`$ neutron meter-label-rule-delete rule-uuid`

Advanced operational features

Logging settings

Networking components use Python logging module to do logging. Logging configuration can be provided in `neutron.conf` or as command-line options. Command options override ones in `neutron.conf`.

To configure logging for Networking components, use one of these methods:

- Provide logging settings in a logging configuration file.

  See Python logging how-to to learn more about logging.

- Provide logging setting in `neutron.conf`

```
[DEFAULT]
# Default log level is WARNING
# Show debugging output in logs (sets DEBUG log level output)
# debug = False

# Show more verbose log output (sets INFO log level output) if debug is
 False
# verbose = False

# log_format = %(asctime)s %(levelname)8s [%(name)s] %(message)s
# log_date_format = %Y-%m-%d %H:%M:%S
# use_syslog = False
# syslog_log_facility = LOG_USER

# if use_syslog is False, we can set log_file and log_dir.
# if use_syslog is False and we do not set log_file,
# the log will be printed to stdout.
# log_file =
# log_dir =
```

Notifications

Notifications can be sent when Networking resources such as network, subnet and port are created, updated or deleted.

Notification options

To support DHCP agent, rpc_notifier driver must be set. To set up the notification, edit notification options in `neutron.conf`:

```
# ============ Notification System Options =====================

# Notifications can be sent when network/subnet/port are create, updated or
 deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = neutron.openstack.common.notifier.no_op_notifier
# Logging driver
# notification_driver = neutron.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = neutron.openstack.common.notifier.rpc_notifier

# default_notification_level is used to form actual topic names or to set
 logging level
# default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma-separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications
```

Setting cases

Logging and RPC

These options configure the Networking server to send notifications through logging and RPC. The logging options are described in *OpenStack Configuration Reference* . RPC notifications go to 'notifications.info' queue bound to a topic exchange defined by 'control_exchange' in `neutron.conf`.

```
# ============ Notification System Options =====================
# Notifications can be sent when network/subnet/port are create, updated or
 deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = neutron.openstack.common.notifier.no_op_notifier
# Logging driver
notification_driver = neutron.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = neutron.openstack.common.notifier.rpc_notifier

# default_notification_level is used to form actual topic names or to set
 logging level
default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications
```

Multiple RPC topics

These options configure the Networking server to send notifications to multiple RPC topics. RPC notifications go to 'notifications_one.info' and 'notifications_two.info' queues bound to a topic exchange defined by 'control_exchange' in `neutron.conf`.

```
# ============ Notification System Options =====================

# Notifications can be sent when network/subnet/port are create, updated or
 deleted.
# There are three methods of sending notifications: logging (via the
# log_file directive), rpc (via a message queue) and
# noop (no notifications sent, the default)

# Notification_driver can be defined multiple times
# Do nothing driver
# notification_driver = neutron.openstack.common.notifier.no_op_notifier
# Logging driver
# notification_driver = neutron.openstack.common.notifier.log_notifier
# RPC driver
notification_driver = neutron.openstack.common.notifier.rabbit_notifier

# default_notification_level is used to form actual topic names or to set
 logging level
default_notification_level = INFO

# default_publisher_id is a part of the notification payload
# host = myhost.com
# default_publisher_id = $host

# Defined in rpc_notifier for rpc way, can be comma separated values.
# The actual topic names will be %s.%(default_notification_level)s
notification_topics = notifications_one,notifications_two
```

Authentication and authorization

Networking uses the Identity Service as the default authentication service. When the Identity Service is enabled, users who submit requests to the Networking service must provide an authentication token in `X-Auth-Token` request header. Users obtain this token by authenticating with the Identity Service endpoint. For more information about authentication with the Identity Service, see *OpenStack Identity Service API v2.0 Reference*. When the Identity Service is enabled, it is not mandatory to specify the tenant ID for resources in create requests because the tenant ID is derived from the authentication token.

Note

The default authorization settings only allow administrative users to create resources on behalf of a different tenant. Networking uses information received from Identity to authorize user requests. Networking handles two kind of authorization policies:

- **Operation-based** policies specify access criteria for specific operations, possibly with fine-grained control over specific attributes;
- **Resource-based** policies specify whether access to specific resource is granted or not according to the permissions configured for the resource (currently available only for the network resource). The actual authorization policies enforced in Networking might vary from deployment to deployment.

The policy engine reads entries from the `policy.json` file. The actual location of this file might vary from distribution to distribution. Entries can be updated while the system is running, and no service restart is required. Every time the policy file is updated, the policies are automatically reloaded. Currently the only way of updating such policies is to edit the policy file. In this section, the terms *policy* and *rule* refer to objects that are specified in the same way in the policy file. There are no syntax differences between a rule and a policy. A policy is something that is matched directly from the Networking policy engine. A rule is an element in a policy, which is evaluated. For instance in `create_subnet: [["admin_or_network_owner"]]`, *create_subnet* is a policy, and *admin_or_network_owner* is a rule.

Policies are triggered by the Networking policy engine whenever one of them matches an Networking API operation or a specific attribute being used in a given operation. For instance the `create_subnet` policy is triggered every time a `POST /v2.0/subnets` request is sent to the Networking server; on the other hand `create_network:shared` is triggered every time the *shared* attribute is explicitly specified (and set to a value different from its default) in a `POST /v2.0/networks` request. It is also worth mentioning that policies can be also related to specific API extensions; for instance `extension:provider_network:set` is be triggered if the attributes defined by the Provider Network extensions are specified in an API request.

An authorization policy can be composed by one or more rules. If more rules are specified, evaluation policy succeeds if any of the rules evaluates successfully; if an API operation matches multiple policies, then all the policies must evaluate successfully. Also, authorization rules are recursive. Once a rule is matched, the rule(s) can be resolved to another rule, until a terminal rule is reached.

The Networking policy engine currently defines the following kinds of terminal rules:

- **Role-based rules** evaluate successfully if the user who submits the request has the specified role. For instance `"role:admin"` is successful if the user who submits the request is an administrator.
- **Field-based rules** evaluate successfully if a field of the resource specified in the current request matches a specific value. For instance `"field:networks:shared=True"` is successful if the `shared` attribute of the `network` resource is set to true.
- **Generic rules** compare an attribute in the resource with an attribute extracted from the user's security credentials and evaluates successfully if the comparison is successful. For instance `"tenant_id:%(tenant_id)s"` is successful if the tenant identifier in the resource is equal to the tenant identifier of the user submitting the request.

This extract is from the default `policy.json` file:

```
{
   "admin_or_owner":[ ❶
      [
         "role:admin"
      ],
      [
         "tenant_id:%(tenant_id)s"
      ]
   ],
   "admin_or_network_owner":[
      [
         "role:admin"
      ],
      [
         "tenant_id:%(network_tenant_id)s"
      ]
   ],
   "admin_only":[
      [
         "role:admin"
      ]
   ],
   "regular_user":[

   ],
   "shared":[
      [
         "field:networks:shared=True"
      ]
   ],
   "default":[ ❷
      [
         "rule:admin_or_owner"
      ]
   ],
   "create_subnet":[
      [
         "rule:admin_or_network_owner"
      ]
   ],
   "get_subnet":[
      [
         "rule:admin_or_owner"
      ],
      [
         "rule:shared"
      ]
   ],
   "update_subnet":[
      [
         "rule:admin_or_network_owner"
      ]
   ],
   "delete_subnet":[
      [
         "rule:admin_or_network_owner"
      ]
   ],
   "create_network":[

   ],
   "get_network":[ ❸
      [
         "rule:admin_or_owner"
      ],
      [
         "rule:shared"
      ]
   ],
   "create_network:shared":[ ❹
      [
         "rule:admin_only"
      ]
   ],
   "update_network":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "delete_network":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "create_port":[

   ],
   "create_port:mac_address":[ ❺
      [
         "rule:admin_or_network_owner"
      ]
   ],
   "create_port:fixed_ips":[
      [
         "rule:admin_or_network_owner"
      ]
   ],
   "get_port":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "update_port":[
      [
         "rule:admin_or_owner"
      ]
   ],
   "delete_port":[
      [
         "rule:admin_or_owner"
      ]
   ]
}
```

❶ A rule that evaluates successfully if the current user is an administrator or the owner of the resource specified in the request (tenant identifier is equal).

❷ The default policy that is always evaluated if an API operation does not match any of the policies in `policy.json`.

❸ This policy evaluates successfully if either *admin_or_owner*, or *shared* evaluates successfully.

❹ This policy restricts the ability to manipulate the *shared* attribute for a network to administrators only.

❺ This policy restricts the ability to manipulate the *mac_address* attribute for a port only to administrators and the owner of the network where the port is attached.

In some cases, some operations are restricted to administrators only. This example shows you how to modify a policy file to permit tenants to define networks and see their resources and permit administrative users to perform all other operations:

```
{
        "admin_or_owner": [["role:admin"], ["tenant_id:%(tenant_id)s"]],
        "admin_only": [["role:admin"]], "regular_user": [],
        "default": [["rule:admin_only"]],
        "create_subnet": [["rule:admin_only"]],
        "get_subnet": [["rule:admin_or_owner"]],
        "update_subnet": [["rule:admin_only"]],
        "delete_subnet": [["rule:admin_only"]],
        "create_network": [],
        "get_network": [["rule:admin_or_owner"]],
        "create_network:shared": [["rule:admin_only"]],
        "update_network": [["rule:admin_or_owner"]],
        "delete_network": [["rule:admin_or_owner"]],
        "create_port": [["rule:admin_only"]],
        "get_port": [["rule:admin_or_owner"]],
        "update_port": [["rule:admin_only"]],
        "delete_port": [["rule:admin_only"]]
        }
```

High availability

The use of high availability in a Networking deployment helps mitigate the impact of individual node failures. In general, you can run `neutron-server` and `neutron-dhcp-agent` in an *active/active* fashion. You can run the `neutron-l3-agent` service as *active/passive*, which avoids IP conflicts with respect to gateway IP addresses.

Networking high availability with Pacemaker

You can run some Networking services in a cluster configuration (active/passive or active/active for Networking server only) with Pacemaker.

Download the latest resources agents:

- neutron-server: https://github.com/madkiss/openstack-resource-agents
- neutron-dhcp-agent: https://github.com/madkiss/openstack-resource-agents
- neutron-l3-agent: https://github.com/madkiss/openstack-resource-agents

Note

For information about how to build a cluster, see Pacemaker documentation.

Plug-in pagination and sorting support

Table 7.42. Plug-ins that support native pagination and sorting

Plug-in	Support Native Pagination	Support Native Sorting
ML2	True	True
Open vSwitch	True	True
Linux Bridge	True	True

Appendix A. Community support

Table of Contents

The following resources are available to help you run and use OpenStack. The OpenStack community constantly improves and adds to the main features of OpenStack, but if you have any questions, do not hesitate to ask. Use the following resources to get OpenStack support, and troubleshoot your installations.

Documentation

For the available OpenStack documentation, see docs.openstack.org.

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

The following books explain how to install an OpenStack cloud and its associated components:

- *Installation Guide for Debian 7.0*
- *Installation Guide for openSUSE and SUSE Linux Enterprise Server*
- *Installation Guide for Red Hat Enterprise Linux, CentOS, and Fedora*
- *Installation Guide for Ubuntu 12.04/14.04 (LTS)*

The following books explain how to configure and run an OpenStack cloud:

- *Cloud Administrator Guide*
- *Configuration Reference*
- *Operations Guide*
- *High Availability Guide*
- *Security Guide*

- *Virtual Machine Image Guide*

The following books explain how to use the OpenStack dashboard and command-line clients:

- *API Quick Start*
- *End User Guide*
- *Admin User Guide*
- *Command-Line Interface Reference*

The following documentation provides reference and guidance information for the OpenStack APIs:

- OpenStack API Complete Reference (HTML)
- API Complete Reference (PDF)
- *OpenStack Block Storage Service API v2 Reference*
- *OpenStack Compute API v2 and Extensions Reference*
- *OpenStack Identity Service API v2.0 Reference*
- *OpenStack Image Service API v2 Reference*
- *OpenStack Networking API v2.0 Reference*
- *OpenStack Object Storage API v1 Reference*

The *Training Guides* offer software training for cloud administration and management.

ask.openstack.org

During the set up or testing of OpenStack, you might have questions about how a specific task is completed or be in a situation where a feature does not work correctly. Use the ask.openstack.org site to ask questions and get answers. When you visit the http://ask.openstack.org site, scan the recently asked questions to see whether your question has already been answered. If not, ask a new question. Be sure to give a clear, concise summary in the title and provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.

OpenStack mailing lists

A great way to get answers and insights is to post your question or problematic scenario to the OpenStack mailing list. You can learn from and help others who might have similar issues. To subscribe or view the archives, go to http://lists.openstack.org/cgi-bin/mailman/listinfo/openstack. You might be interested in the other mailing lists for specific projects or development, which you can find on the wiki. A description of all mailing lists is available at http://wiki.openstack.org/MailingLists.

The OpenStack wiki

The OpenStack wiki contains a broad range of topics but some of the information can be difficult to find or is a few pages deep. Fortunately, the wiki search feature enables you to search by title or content. If you search for specific information, such as about networking or nova, you can find a large amount of relevant material. More is being added all the time, so be sure to check back often. You can find the search box in the upper-right corner of any OpenStack wiki page.

The Launchpad Bugs area

The OpenStack community values your set up and testing efforts and wants your feedback. To log a bug, you must sign up for a Launchpad account at https://launchpad.net/+login. You can view existing bugs and report bugs in the Launchpad Bugs area. Use the search feature to determine whether the bug has already been reported or already been fixed. If it still seems like your bug is unreported, fill out a bug report.

Some tips:

- Give a clear, concise summary.
- Provide as much detail as possible in the description. Paste in your command output or stack traces, links to screen shots, and any other information which might be useful.
- Be sure to include the software and package versions that you are using, especially if you are using a development branch, such as, `"Juno release" vs git commit bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment-specific information is helpful, such as whether you are using Ubuntu 14.04 or are performing a multi-node installation.

The following Launchpad Bugs areas are available:

- Bugs: OpenStack Block Storage (cinder)
- Bugs: OpenStack Compute (nova)
- Bugs: OpenStack Dashboard (horizon)
- Bugs: OpenStack Identity (keystone)
- Bugs: OpenStack Image Service (glance)
- Bugs: OpenStack Networking (neutron)
- Bugs: OpenStack Object Storage (swift)
- Bugs: Bare Metal (ironic)
- Bugs: Data Processing Service (sahara)
- Bugs: Database Service (trove)

- Bugs: Orchestration (heat)
- Bugs: Telemetry (ceilometer)
- Bugs: Queue Service (marconi)
- Bugs: OpenStack API Documentation (api.openstack.org)
- Bugs: OpenStack Documentation (docs.openstack.org)

The OpenStack IRC channel

The OpenStack community lives in the #openstack IRC channel on the Freenode network. You can hang out, ask questions, or get immediate feedback for urgent and pressing issues. To install an IRC client or use a browser-based client, go to http://webchat.freenode.net/. You can also use Colloquy (Mac OS X, http://colloquy.info/), mIRC (Windows, http://www.mirc.com/), or XChat (Linux). When you are in the IRC channel and want to share code or command output, the generally accepted method is to use a Paste Bin. The OpenStack project has one at http://paste.openstack.org. Just paste your longer amounts of text or logs in the web form and you get a URL that you can paste into the channel. The OpenStack IRC channel is `#openstack` on `irc.freenode.net`. You can find a list of all OpenStack IRC channels at https://wiki.openstack.org/wiki/IRC.

Documentation feedback

To provide feedback on documentation, join and use the `<openstack-docs@lists.openstack.org>` mailing list at OpenStack Documentation Mailing List, or report a bug.

OpenStack distribution packages

The following Linux distributions provide community-supported packages for OpenStack:

- **Debian:** http://wiki.debian.org/OpenStack
- **CentOS, Fedora, and Red Hat Enterprise Linux:** http://openstack.redhat.com/
- **openSUSE and SUSE Linux Enterprise Server:** http://en.opensuse.org/Portal:OpenStack
- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive