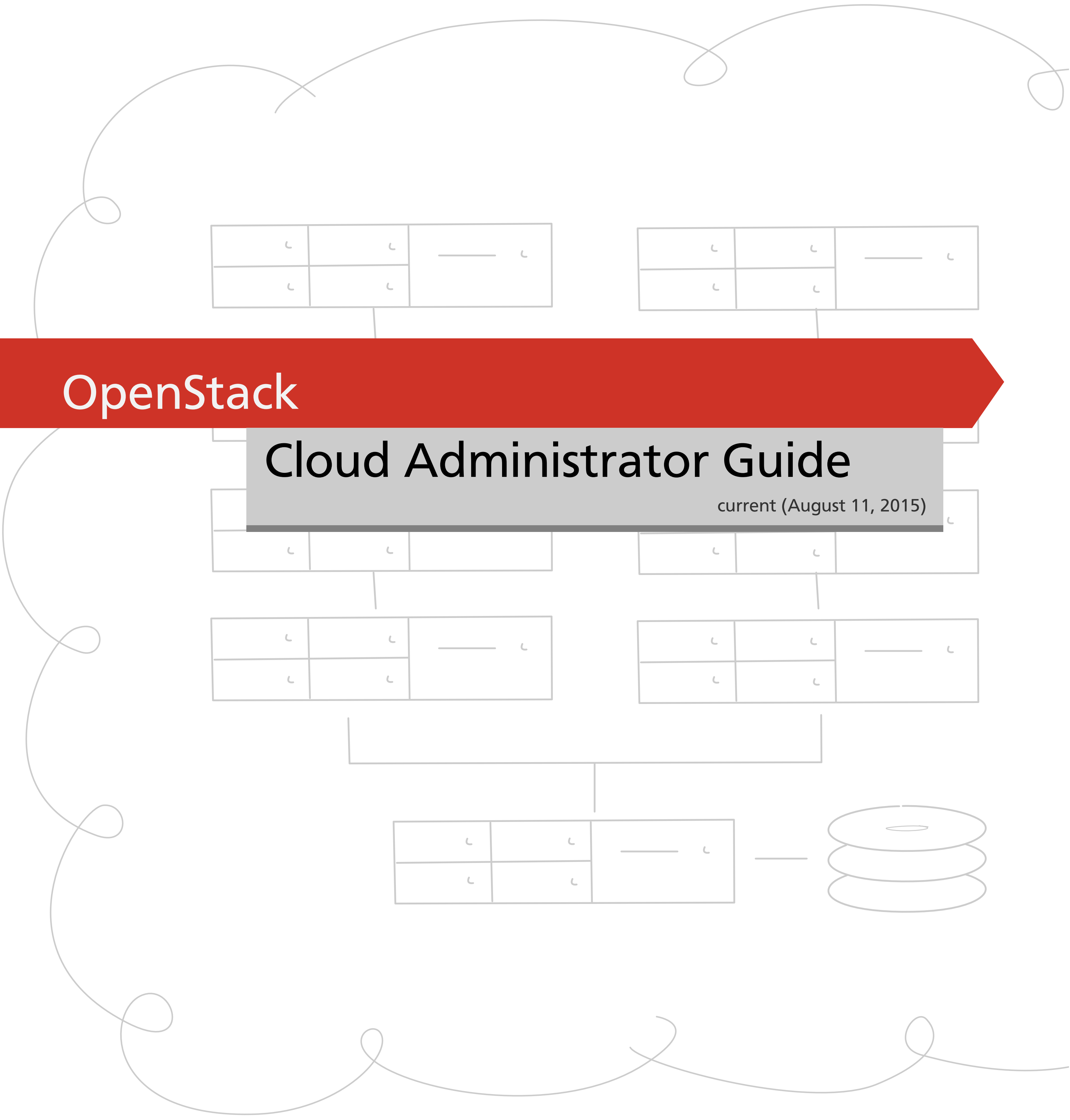
OpenStack

Cloud Administrator Guide

current (August 11, 2015)



docs.openstack.org

OpenStack Cloud Administrator Guide

current (2015-08-11)

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

This guide documents OpenStack Kilo, OpenStack Juno, and OpenStack Icehouse releases.

Table of Contents

List of Figures

List of Tables

List of Examples

Preface

Conventions

The OpenStack documentation uses several typesetting conventions.

Notices

Notices take these forms:

Note

A handy tip or reminder.

Important

Something you must be aware of before proceeding.

Warning

Critical information about the risk of data loss or security issues.

Command prompts

$ prompt	Any user, including the `root` user, can run commands that are prefixed with the $ prompt.
# prompt	The `root` user must run commands that are prefixed with the # prompt. You can also prefix these commands with the **sudo** command, if available, to run them.

Document change history

This version of the guide replaces and obsoletes all earlier versions.

The following table describes the most recent changes:

Revision Date	Summary of Changes
February 20, 2015	• For the Kilo release, the guide has been updated with a new Measurements section in the Telemetry chapter. The tables contain the release information for all collected meters regarding to when they were introduced in the module. In addition, the Orchestration chapter has been added to the guide. It describes in details Orchestration module available in OpenStack since Havana release.
October 15, 2014	• For the Juno release, the guide has been updated with a new Telemetry chapter.
July 21, 2014	• Updated variables to use correct formatting.
April 17, 2014	• For the Icehouse release, the guide was organized with system administration and system architecture sections. Also, how-to sections were moved to this guide instead of the *OpenStack Configuration Reference*.
November 12, 2013	• Adds options for tuning operational status synchronization in the NSX plug-in.

Revision Date	Summary of Changes
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

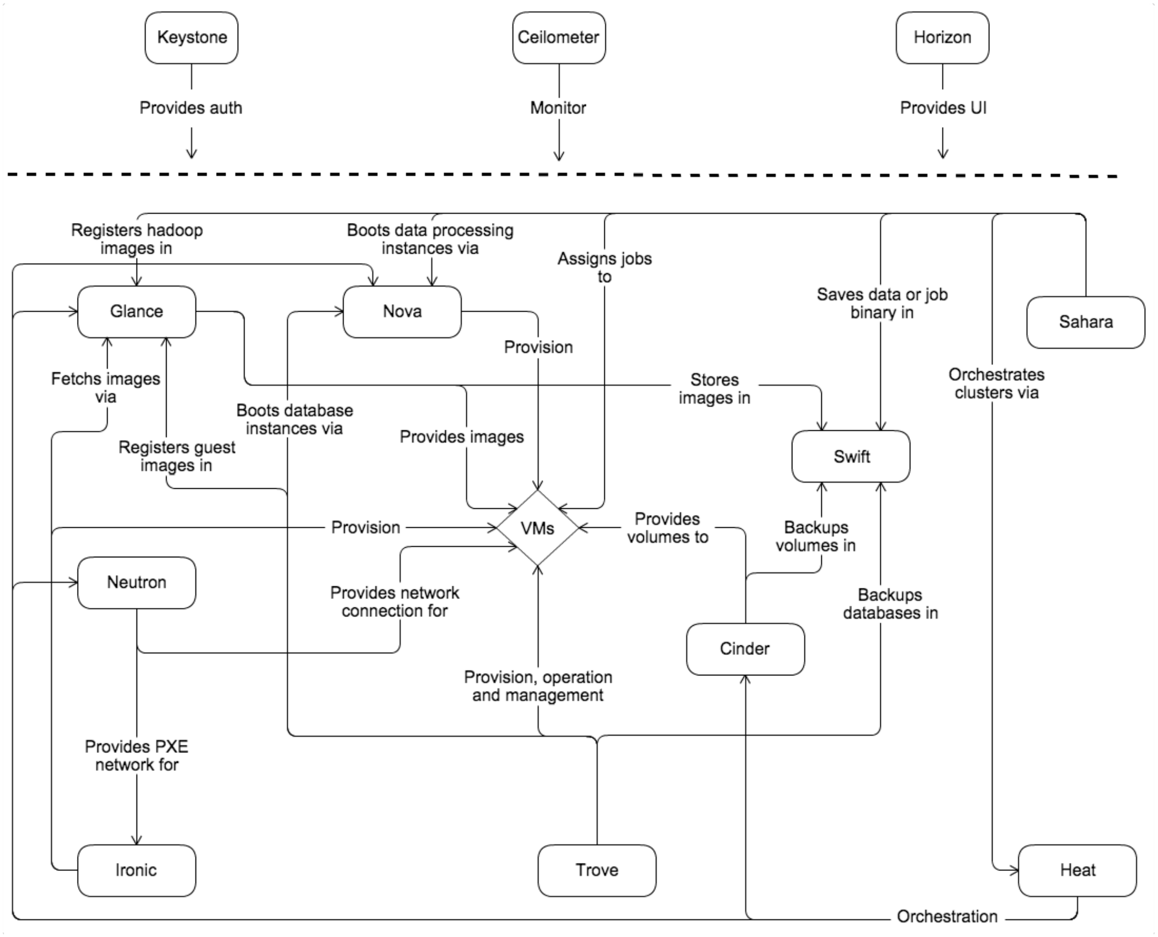
Service	Project name	Description
Dashboard	*Horizon*	Provides a web-based self-service portal to interact with underlying OpenStack services, such as launching an instance, assigning IP addresses and configuring access controls.
Compute	*Nova*	Manages the lifecycle of compute instances in an OpenStack environment. Responsibilities include spawning, scheduling and decommissioning 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. In this case, it writes objects and files to multiple drives, ensuring the data is replicated across a server cluster.
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.

Service	Project name	Description
	Higher-level services	
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.
Data processing service	*Sahara*	Provides capabilties to provision and scale Hadoop clusters in OpenStack by specifying parameters like Hadoop version, cluster topology and nodes hardware details.

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.

As shown in Figure 1.1, “OpenStack conceptual architecture” [2], OpenStack consists of several independent parts, named the OpenStack services. All services authenticate through a common Identity service. Individual services interact with each other through public APIs, except where privileged administrator commands are necessary.

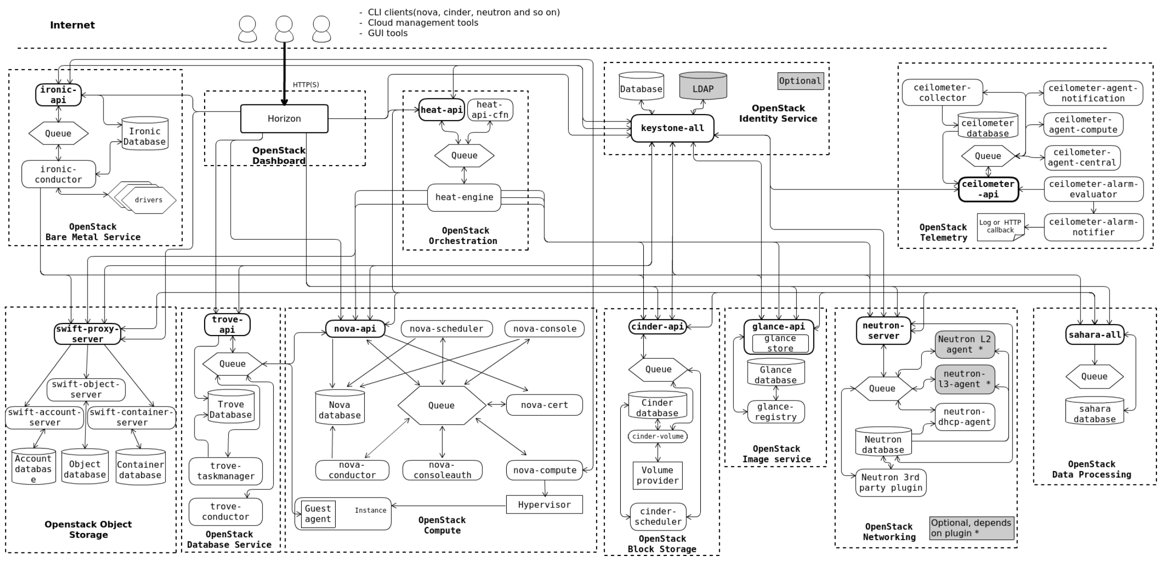
Internally, OpenStack services are composed of several processes. All services have at least one API process, which listens for API requests, preprocesses them and passes them on to other parts of the service. With the exception of the Identity service, the actual work is done by distinct processes.

For communication between the processes of one service, an AMQP *message broker* is used. The service's state is stored in a database. When deploying and configuring your OpenStack cloud, you can choose among several message broker and database solutions, such as RabbitMQ, Qpid, MySQL, MariaDB, and SQLite.

Users can access OpenStack via the web-based user interface implemented by the dashboard service, via command-line clients and by issuing API requests through tools like browser plug-ins or **curl**. For applications, several SDKs are available. Ultimately, all these access methods issue REST API calls to the various OpenStack services.

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

Figure 1.2. Logical architecture



OpenStack services

This section describes OpenStack services in detail.

OpenStack Compute

Use OpenStack Compute to host and manage cloud computing systems. OpenStack Compute is a major part of an Infrastructure-as-a-Service (IaaS) system. The main modules are implemented in Python.

OpenStack Compute interacts with OpenStack Identity for authentication, OpenStack Image service for disk and server images, and OpenStack dashboard for the user and administrative interface. Image access is limited by projects, and by users; quotas are limited per project (the number of instances, for example). OpenStack Compute can scale horizontally on standard hardware, and download images to launch instances.

OpenStack Compute consists of the following areas and their components:

API

`nova-api` service

Accepts and responds to end user compute API calls. The service supports the OpenStack Compute API, the Amazon EC2 API, and a special Admin API for privileged users to perform administrative actions. It enforces some policies and initiates most orchestration activities, such as running an instance.

`nova-api-metadata` service

Accepts metadata requests from instances. The `nova-api-metadata` service is generally used when you run in multi-host mode with `nova-network` installations. For details, see Metadata service in the *OpenStack Cloud Administrator Guide*.

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

Compute core

`nova-compute` service

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

Processing is fairly complex. Basically, the daemon accepts actions from the queue and performs a series of system commands such as launching a KVM instance and updating its state in the database.

`nova-scheduler` service	Takes a virtual machine instance request from the queue and determines on which compute server host it runs.
`nova-conductor` module	Mediates interactions between the `nova-compute` service and the database. It eliminates direct accesses to the cloud database made by the `nova-compute` service. The `nova-conductor` module scales horizontally. However, do not deploy it on nodes where the `nova-compute` service runs. For more information, see A new Nova service: nova-conductor.
`nova-cert` module	A server daemon that serves the Nova Cert service for X509 certificates. Used to generate certificates for **euca-bundle-image**. Only needed for the EC2 API.

Networking for VMs

`nova-network` worker daemon	Similar to the `nova-compute` service, accepts networking tasks from the queue and manipulates the network. Performs tasks such as setting up bridging interfaces or changing IPtables rules.

Console interface

`nova-consoleauth` daemon	Authorizes tokens for users that console proxies provide. See `nova-novncproxy` and `nova-xvpvncproxy`. This service must be running for console proxies to work. You can run proxies of either type 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-spicehtml5proxy` daemon	Provides a proxy for accessing running instances through a SPICE connection. Supports browser-based HTML5 client.
`nova-xvpvncproxy` daemon	Provides a proxy for accessing running instances through a VNC connection. Supports an OpenStack-specific Java client.
`nova-cert` daemon	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	An S3 interface for registering images with the OpenStack Image service. Used primarily 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. Although it is 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.

Other components

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

SQL database Stores most build-time and run-time states for a cloud infrastructure, including:

- Available instance types
- Instances in use
- Available networks
- Projects

Theoretically, OpenStack Compute can support any database that SQL-Alchemy supports. Common databases are SQLite3 for test and development work, MySQL, and PostgreSQL.

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
Encryption is available	Encryption is available	Work in progress - Expected for the Mitaka release
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

You should note that:

- *You cannot use OpenStack Object Storage like a traditional hard drive.* The Object Storage relaxes some of the constraints of a POSIX-style file system to get other gains. You can access the objects through an API which uses HTTP. Subsequently you don't have to provide atomic operations (that is, relying on eventual consistency), you can scale a storage system easily 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.* It provides an abstraction to different methods for storage - a bridge to the storage, not the storage itself.
- *The OpenStack Object Storage can function on its own.* The Object Storage (swift) product can be used independently of the Compute (nova) product.

Command-line clients and other interfaces

swift client	Enables users to submit commands to the REST API through a command-line client authorized as either a admin user, reseller user, or swift user.
swift-init	Script that initializes the building of the ring file, takes daemon names as parameter and offers commands. Documented in http://docs.openstack.org/developer/swift/admin_guide.html#managing-services.
swift-recon	
swift-ring-builder	Storage ring build and rebalance utility. Documented in http://docs.openstack.org/developer/swift/admin_guide.html#managing-the-rings.

OpenStack Object Storage

The OpenStack Object Storage is a multi-tenant object storage system. It is highly scalable and can manage large amounts of unstructured data at low cost through a RESTful HTTP API.

It includes the following components:

Proxy servers (`swift-proxy-server`)	Accepts OpenStack 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 that is usually deployed with memcache.
Account servers (`swift-account-server`)	Manages accounts defined with Object Storage.
Container servers (`swift-container-server`)	Manages the mapping of containers or folders, within Object Storage.
Object servers (`swift-object-server`)	Manages actual objects,such as files, on the storage nodes.
Various 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.
WSGI middleware	Handles authentication and is usually OpenStack Identity.

OpenStack Block Storage

The OpenStack Block Storage service (cinder) adds persistent storage to a virtual machine. Block Storage provides an infrastructure for managing volumes, and interacts with OpenStack Compute to provide volumes for instances. The service also enables management of volume snapshots, and volume types.

The Block Storage service consists of the following components:

`cinder-api`	Accepts API requests, and routes them to the `cinder-volume` for action.
`cinder-volume`	Interacts directly with the Block Storage service, and processes such as the `cinder-scheduler`. It also interacts with these processes through a message queue. The `cinder-volume service` responds to read and write requests sent to the Block Storage service to maintain state. It can interact with a variety of storage providers through a driver architecture.

`cinder-scheduler` daemon	Selects the optimal storage provider node on which to create the volume. A similar component to the `nova-scheduler`.
`cinder-backup` daemon	The `cinder-backup` service provides backing up volumes of any type to a backup storage provider. Like the `cinder-volume` service, it can interact with a variety of storage providers through a driver architecture.
Messaging queue	Routes information between the Block Storage processes.

OpenStack Networking

OpenStack Networking allows you to create and attach interface devices managed by other OpenStack services to networks. Plug-ins can be implemented to accommodate different networking equipment and software, providing flexibility to OpenStack architecture and deployment.

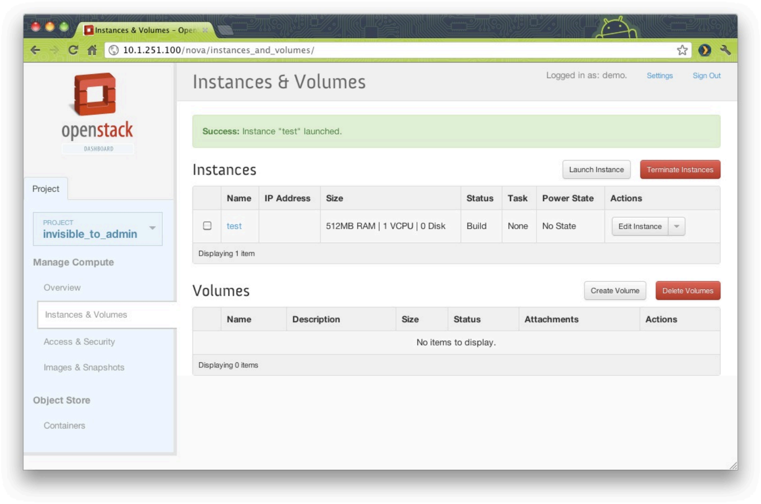
It includes the following components:

`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, and the VMware NSX product. The common agents are L3 (layer 3), DHCP (dynamic host IP addressing), and a plug-in agent.
Messaging queue	Used by most OpenStack Networking installations 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 mainly interacts with OpenStack Compute to provide networks and connectivity for its instances.

OpenStack dashboard

The OpenStack 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.

OpenStack Identity

The OpenStack *Identity Service* performs the following functions:

- Tracking users and their permissions.
- Providing a catalog of available services with their API endpoints.

When installing OpenStack Identity service, you must register each service in your OpenStack installation. Identity service can then track which OpenStack services are installed, and where they are located on the network.

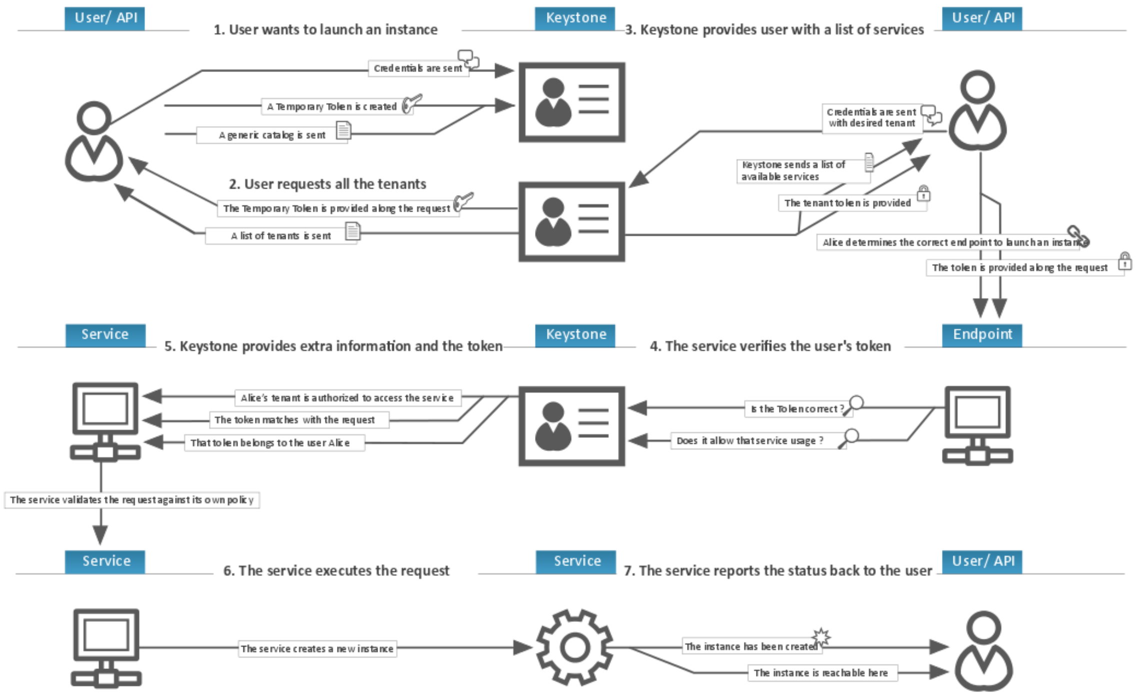
To understand OpenStack Identity, you must understand the following concepts:

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 confirms the user's identity. For example: user name and password, user name and API key, or an authentication token provided by the Identity Service.

Authentication	The process of confirming the identity of a user. OpenStack Identity 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. When user credentials are validated, OpenStack Identity issues an authentication token which the user provides in subsequent requests.
Token	An alpha-numeric string of text used to access OpenStack APIs and resources. A token may be revoked at any time and is valid for a finite duration. While OpenStack Identity supports token-based authentication in this release, the intention is to support additional protocols in the future. Its main purpose is to be an integration service, and not aspire to be a full-fledged identity store and management solution.
Tenant	A container used to group or isolate resources. Tenants also group or isolate 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). It provides one or more endpoints in which users can access resources and perform operations.
Endpoint	A network-accessible address where you access a service, usually a URL address. If you are using an extension for templates, an endpoint template can be created, which represents the templates of all the consumable services that are available across the regions.
Role	A personality with a defined set of user rights and privileges to perform a specific set of operations. In the Identity service, a token that is issued to a user includes the list of roles. 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.
Keystone Client	A command line interface for the OpenStack Identity API. For example, users can run the **keystone service-create** and **keystone endpoint-create** commands to register services in their OpenStack installations.

The following diagram shows the OpenStack Identity process flow:

The Keystone Identity Manager


User/ API
1. User wants to launch an instance
Keystone
3. Keystone provides user with a list of services
User/ API
Credentials are sent
A Temporary Token is created
A generic catalog is sent
Credentials are sent with desired tenant
Keystone sends a list of available services
The tenant token is provided
2. User requests all the tenants
The Temporary Token is provided along the request
A list of tenants is sent
Alice determines the correct end point to launch an instance
The token is provided along the request
Service
5. Keystone provides extra information and the token
Keystone
4. The service verifies the user'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 the user
User/ API
The service creates a new instance
The instance has been created
The instance is reachable here

OpenStack Image service

The OpenStack Image service is central to Infrastructure-as-a-Service (IaaS) as shown in the section called "Conceptual architecture" [2]. It accepts API requests for disk or server images, and image metadata from end users or OpenStack Compute components. It also supports the storage of disk or server images on various repository types, including OpenStack Object Storage.

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

The OpenStack 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 for use by OpenStack Image service. Do not disclose it to users.
Database	Stores image metadata and you can choose your database depending on your preference. Most deployments use MySQL or SQLite.
Storage repository for image files	Various repository types are supported including normal file systems, Object Storage, RADOS block devices, HTTP, and Amazon S3. Note that some repositories will only support read-only usage.

OpenStack Telemetry module

The Telemetry module performs the following functions:

- Efficiently polls metering data related to OpenStack services.
- Collects event and metering data by monitoring notifications sent from services.
- Publishes collected data to various targets including data stores and message queues.
- Creates alarms when collected data breaks defined rules.

The Telemetry module consists of the following 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 our focus is 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. Multiple agents can be started to scale service horizontally.
A notification agent (`ceilometer-agent-notification`)	Runs on a central management server(s) and consumes messages from the message queue(s) to build event and metering data.
A collector (`ceilometer-collector`)	Runs on central management server(s) and dispatches collected telemetry data to a data store or external consumer without modification.
An alarm evaluator (`ceilometer-alarm-evaluator`)	Runs on one or more central management servers to determine when alarms fire due to the associated statistic trend crossing a threshold over a sliding time window.
An alarm notifier (`ceilometer-alarm-notifier`)	Runs on one or more central management servers to allow alarms to be set based on the threshold evaluation for a collection of samples.
An API server (`ceilometer-api`)	Runs on one or more central management servers to provide data access from the data store.

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

OpenStack Orchestration module

The Orchestration module 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 allow you to create most OpenStack resource types, such as instances, floating IPs, volumes, security groups and users. It also provides advanced functionality, such as instance high availability, instance auto-scaling, and nested stacks. This enables OpenStack core projects to receive a larger user base.

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

The Orchestration module consists of the following components:

`heat` command-line client	A CLI that communicates with the heat-api to run OpenStack-native APIs.
`heat-api` component	An OpenStack-native REST API that processes API requests by sending them to the heat-engine over Remote Procedure Call (RPC).
`heat-api-cfn` component	An AWS Query API that is compatible with AWS CloudFormation. It 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.

OpenStack Database service

The Database service provides scalable and reliable cloud provisioning functionality for both relational and non-relational database engines. Users can quickly and easily use database features without the burden of handling complex administrative tasks. Cloud users and database administrators can provision and manage multiple database instances as needed.

The Database service provides resource isolation at high performance levels, and automates complex administrative tasks such as deployment, configuration, patching, backups, restores, and monitoring.

Process flow example. This example is a high-level process flow for using Database services:

1. The OpenStack Administrator configures the basic infrastructure using the following steps:

   a. Install the Database service.

   b. Create an image for each type of database. For example, one for MySQL and one for MongoDB.

   c. Use the **trove-manage** command to import images and offer them to tenants.

2. The OpenStack end user deploys the Database service using the following steps:

   a. Create a Database service instance using the **trove create** command.

   b. Use the **trove list** command to get the ID of the instance, followed by the **trove show** command to get the IP address of it.

   c. Access the Database service instance using typical database access commands. For example, with MySQL:

   ```
   $ mysql -u myuser -p -h TROVE_IP_ADDRESS mydb
   ```

The Database service includes the following components:

`python-troveclient` command-line client	A CLI that communicates with the `trove-api` component.
`trove-api` component	Provides an OpenStack-native RESTful API that supports JSON to provision and manage Trove instances.
`trove-conductor` service	Runs on the host, and receives messages from guest instances that want to update information on the host.
`trove-taskmanager` service	Instruments the complex system flows that support provisioning instances, managing the lifecycle of instances, and performing operations on instances.

`trove-guestagent` service	Runs within the guest instance. Manages and performs operations on the database itself.

OpenStack Data processing service

The Data processing service for OpenStack (sahara) aims to provide users with a simple means to provision data processing (Hadoop, Spark) clusters by specifying several parameters like Hadoop version, cluster topology, node hardware details and a few more. After a user fills in all the parameters, the Data processing service deploys the cluster in a few minutes. Sahara also provides a means to scale already provisioned clusters by adding/removing worker nodes on demand.

The solution addresses the following use cases:

- Fast provisioning of Hadoop clusters on OpenStack for development and QA.
- Utilization of unused compute power from general purpose OpenStack IaaS cloud.
- Analytics-as-a-Service for ad-hoc or bursty analytic workloads.

Key features are:

- Designed as an OpenStack component.
- Managed through REST API with UI available as part of OpenStack dashboard.
- Support for different Hadoop distributions:
  - Pluggable system of Hadoop installation engines.
  - Integration with vendor specific management tools, such as Apache Ambari or Cloudera Management Console.
- Predefined templates of Hadoop configurations with the ability to modify parameters.
- User-friendly UI for ad-hoc analytics queries based on Hive or Pig.

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

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

Identity 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`:

  ```
  $ openstack user create --password-prompt --email alice@example.com alice
  ```

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

  ```
  $ openstack project create acme
  ```

- **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 projects and users. Users may be given a domain's administrator role. A domain administrator may create projects, users, and groups within a domain and assign roles to users and groups.

This example creates a domain named `emea`:

```
$ openstack --os-identity-api-version=3 domain create emea
```

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

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

```
$ openstack role create 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:

```
$ openstack user list
+--------+-------+
| ID     | Name  |
+--------+-------+
| 892585 | alice |
+--------+-------+
```

```
$ openstack role list
+--------+---------------+
| ID     | Name          |
+--------+---------------+
| 9a764e | compute-user  |
+--------+---------------+
```

```
$ openstack project list
+--------+--------------------+
| ID     | Name               |
+--------+--------------------+
| 6b8fd2 | acme               |
+--------+--------------------+
```

```
$ openstack role add --project 6b8fd2 --user 892585 9a764e
```

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 or 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 Web Server Gateway Interface (WSGI) service`. Can be run in a WSGI-capable web server such as Apache httpd to provide the Identity Service. The service and administrative APIs are run as separate instances of the WSGI service.

- 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.
- `keystone-all`. Starts both the service and administrative APIs in a single process. Using federation with keystone-all is not supported. keystone-all is deprecated in favor of the WSGI service.

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.

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 a 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 **keystone-manage** as root, you can append `--keystone-user` and `--keystone-group` parameters to set the user name 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:

- `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`.
- `ca_key` - Location of the private key used by the CA. Default is `/etc/keystone/ssl/private/cakey.pem`.
- `key_size` - Default is `2048`.
- `valid_days` - Default is `3650`.
- `cert_subject` - Certificate subject (auto generated certificate) for token signing. Default is `/C=US/ST=Unset/L=Unset/O=Unset/CN=www.example.com`.

When generating certificates with the **keystone-manage pki_setup** command, the `ca_key`, `key_size`, and `valid_days` configuration options are used.

If the **keystone-manage pki_setup** command is not used to generate certificates, or you are providing your own certificates, these values do not need to be set.

If `provider=keystone.token.providers.uuid.Provider` in the `[token]` section of the keystone configuration, a typical token looks like `53f7f6ef0cc344b5be706bcc8b1479e1`. If `provider=keystone.token.providers.pki.Provider`, 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 **key-stone-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 a 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            = 4096
default_keyfile         = keystonekey.pem
default_md              = sha256

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.

Note

The procedure of copying the key and cert files may be improved if done after first running **keystone-manage pki_setup** since this command also creates other needed files, such as the `index.txt` and `serial` files.

Also, when copying the necessary files to a different server for replicating the functionality, the entire directory of files is needed, not just the key and cert files.

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.

Switching out expired signing certificates

The following procedure details how to switch out expired signing certificates with no cloud outages.

1. Generate a new signing key.
2. Generate a new certificate request.
3. Sign the new certificate with the existing CA to generate a new `signing_cert`.
4. Append the new `signing_cert` to the old `signing_cert`. Ensure the old certificate is in the file first.
5. Remove all signing certificates from all your hosts to force OpenStack Compute to download the new `signing_cert`.
6. Replace the old signing key with the new signing key. Move the new signing certificate above the old certificate in the `signing_cert` file.
7. After the old certificate reads as expired, you can safely remove the old signing certificate from the 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.

Client authentication with keystone-all

When running keystone-all, the server can be configured to enable SSL with client authentication using the following instructions. Modify the `[eventlet_server_ssl]` section in the `etc/keystone.conf` file. The following SSL configuration example uses the included sample certificates:

```
[eventlet_server_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.

When running the Identity Service as a WSGI service in a web server such as Apache httpd, this configuration is done in the web server instead. In this case the options in the `[eventlet_server_ssl]` section are ignored.

External authentication with Identity

When Identity 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 and Kerberos, instead of using the user name and password combination.

Use HTTPD authentication

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

The Openstack Identity Service supports integration with existing LDAP directories for authentication and authorization services.

When the Openstack Identity service is configured to use LDAP back ends, you can split authentication (using the *identity* feature) and authorization (using the *assignment* feature).

The identity feature enables administrators to manage users and groups by each domain or the Openstack Identity Service entirely.

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

Important

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

```
# setsebool -P authlogin_nsswitch_use_ldap on
```

Identity configuration is split into two separate back ends: identity (back end for users and groups) and assignments (back end for domains, projects, roles, role assignments). To configure identity, set options in the `/etc/keystone/keystone.conf` file. See the section called “Integrate identity back end with LDAP” [32] for identity back end configuration examples and the section called “Integrate assignment back end with LDAP” [36] for assignment back end configuration examples. Modify these examples as needed.

Note

Multiple back ends are supported. You can integrate the Openstack Identity service with a single LDAP server (configure both identity and assignments to LDAP, or set identity and assignments back end with SQL or LDAP), or multiple back ends using domain-specific configuration files.

To define the destination LDAP server. 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
```

Note

Configure `dumb_member` if you set `use_dumb_member` to true.

```
[ldap]
dumb_member = cn=dumb,dc=nonexistent
```

Additional LDAP integration settings. Set these options in the `/etc/keystone/keystone.conf` file for a single LDAP server, or `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` files for multiple back ends.

Query option

Use `query_scope` to control the scope level of data presented (search only the first level or search an entire sub-tree) through LDAP.

Use `page_size` to control the maximum results per page. A value of zero disables paging.

Use `alias_dereferencing` to control the LDAP dereferencing option for queries.

Use `chase_referrals` to override the system's default referral chasing behavior for queries.

```
[ldap]
query_scope = sub
page_size = 0
alias_dereferencing = default
chase_referrals =
```

Debug

Use `debug_level` to set the LDAP debugging level for LDAP calls. A value of zero means that debugging is not enabled.

```
[ldap]
debug_level = 0
```

Warning

This value is a bitmask, consult your LDAP documentation for possible values.

Connection pooling

Use `use_pool` to enable LDAP connection pooling. Configure connection pool size, maximum retry, reconnect trials, timeout (-1 indicates indefinite wait) and lifetime in seconds.

```
[ldap]
use_pool = true
pool_size = 10
pool_retry_max = 3
pool_retry_delay = 0.1
pool_connection_timeout = -1
pool_connection_lifetime = 600
```

Connection pooling for end user authentication

Use `use_auth_pool` to enable LDAP connection pooling for end user authentication. Configure connection pool size and lifetime in seconds.

```
[ldap]
use_auth_pool = false
auth_pool_size = 100
auth_pool_connection_lifetime = 60
```

When you have finished configuration, restart the Openstack Identity service:

```
# service keystone restart
```

Warning

During service restart, authentication and authorization are unavailable.

Integrate identity back end with LDAP

The identity back end contains information for users, groups, and group member lists. Integrating the identity back end with LDAP allows administrators to use users and groups in LDAP.

Important

For OpenStack Identity Service to access LDAP servers, you must define the destination LDAP server in the `keystone.conf` file. For more information, see the section called "Integrate Identity with LDAP" [30].

Integrating an identity back end with LDAP

1. Enable the LDAP identity driver in the `keystone.conf` file. This allows LDAP as an identity back end:

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

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

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

   group_tree_dn = ou=Groups,dc=example,dc=org
   group_objectclass = groupOfNames
   ```

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
```

3. 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

group_allow_create = False
group_allow_update = False
group_allow_delete = False
```

4. Restart the OpenStack Identity service:

```
# service keystone restart
```

Warning

During service restart, authentication and authorization are unavailable.

Integrating identity with multiple back ends

1. Set the following options in the `/etc/keystone/keystone.conf` file:

   a. Enable the LDAP driver:

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

   b. Enable domain-specific drivers:

   ```
   [identity]
   domain_specific_drivers_enabled = True
   domain_config_dir = /etc/keystone/domains
   ```

2. Restart the service:

```
# service keystone restart
```

3. List the domains using the dashboard, or the OpenStackClient CLI. Refer to the Command List for a list of OpenStackClient commands.

4. Create domains using OpenStack dashboard, or the OpenStackClient CLI.

5. For each domain, create a domain-specific configuration file in the `/etc/keystone/domains` directory. Use the file naming convention `keystone.`*`DOMAIN_NAME`*`.conf`, where *`DOMAIN_NAME`* is the domain name assigned in the previous step.

Note

The options set in the `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` file will override options in the `/etc/keystone/keystone.conf` file.

6. Define the destination LDAP server in the `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` file. For example:

```
[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
```

7. Create the organizational units (OU) in the LDAP directories, and define their corresponding locations in the `/etc/keystone/domains/keystone.DOMAIN_NAME.conf` file. For example:

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

group_tree_dn = ou=Groups,dc=example,dc=org
group_objectclass = groupOfNames
```

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
```

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

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

group_allow_create = False
group_allow_update = False
group_allow_delete = False
```

9. Restart the OpenStack Identity service:

```
# service keystone restart
```

Warning

During service restart, authentication and authorization are unavailable.

Additional LDAP integration settings. Set these options in the `/etc/keystone/keystone.conf` file for a single LDAP server, or `/etc/keystone/domains/keystone. DOMAIN_NAME.conf` files for multiple back ends.

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)
group_filter =
```

Identity attribute mapping

Mask account status values (include any additional attribute mappings) for compatibility with various directory services. Superfluous accounts are filtered with `user_filter`.

Setting attribute ignore to list of attributes stripped off on update.

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

```
[ldap]
user_id_attribute      = cn
user_name_attribute    = sn
user_mail_attribute    = mail
user_pass_attribute    = userPassword
user_enabled_attribute = userAccountControl
user_enabled_mask      = 2
user_enabled_invert    = false
user_enabled_default   = 51
user_default_project_id_attribute =
user_attribute_ignore = default_project_id,
tenants
user_additional_attribute_mapping =

group_id_attribute     = cn
group_name_attribute   = ou
group_member_attribute = member
group_desc_attribute   = description
group_attribute_ignore =
group_additional_attribute_mapping =
```

Enabled emulation

An alternative method to determine if a user is enabled or not is by checking if that user is a member of the emulation group.

Use DN of the group entry to hold enabled user when using enabled emulation.

```
[ldap]
user_enabled_emulation = false
user_enabled_emulation_dn = false
```

Integrate assignment back end with LDAP

When you configure the OpenStack Identity service to use LDAP servers, you can split authentication and authorization using the *assignment* feature. Integrating the assignment back end with LDAP allows administrators to use projects (tenant), roles, domains, and role assignments in LDAP.

Note

Using LDAP as an assignment back end is not recommended.

Note

The OpenStack Identity service does not support domain-specific assignment back ends.

Important

For OpenStack Identity assignments to access LDAP servers, you must define the destination LDAP server in the `keystone.conf` file. For more information, see the section called "Integrate Identity with LDAP" [30].

Integrating assignment back ends with LDAP

1. Enable the assignment driver. In the `[assignment]` section, set the `driver` configuration key to `keystone.assignment.backends.sql.Assignment`:

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

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

```
[ldap]
role_tree_dn =
role_objectclass = inetOrgPerson

project_tree_dn = ou=Groups,dc=example,dc=org
project_objectclass = groupOfNames
```

Note

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

```
project_objectclass = groupOfNames
```

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

```
[ldap]
role_allow_create = False
role_allow_update = False
role_allow_delete = False

project_allow_create = False
project_allow_update = False
project_allow_delete = False
```

4. Restart the OpenStack Identity service:

```
# service keystone restart
```

Warning

During service restart, authentication and authorization are unavailable.

Additional LDAP integration settings. Set these options in the `/etc/keystone/keystone.conf` file for a single LDAP server, or `/etc/keystone/domains/keystone.` *`DOMAIN_NAME`*`.conf` files for multiple back ends.

Filters

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

```
[ldap]
project_filter = (member=cn=openstack-user,ou=
workgroups,dc=example,dc=org)
role_filter =
```

Warning

Filtering method

Assignment attribute mapping

Mask account status values (include any additional attribute mappings) for compatibility with various directory services. Superfluous accounts are filtered with `user_filter`.

Setting attribute ignore to list of attributes stripped off on update.

```
[ldap]
role_id_attribute = cn
role_name_attribute = ou
role_member_attribute = roleOccupant
role_additional_attribute_mapping =
role_attribute_ignore =

project_id_attribute = cn
project_name_attribute = ou
project_member_attribute = member
project_desc_attribute = description
project_enabled_attribute = enabled
project_domain_id_attribute = businessCategory
project_additional_attribute_mapping =
project_attribute_ignore =
```

Enabled emulation

An alternative method to determine if a project is enabled or not is to check if that project is a member of the emulation group.

Use DN of the group entry to hold enabled projects when using enabled emulation.

```
[ldap]
project_enabled_emulation = false
project_enabled_emulation_dn = false
```

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” [25].

Once you verify the location of your certificate authority file:

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 or X.509 certificate, 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
```

or

```
[token]
bind = x509
```

Currently `kerberos` and `x509` are 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.

- `kerberos`

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

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

- `x509`

  Enables bind checking. Requires use of X.509 as the authentication mechanism for tokens:

  ```
  [token]
  enforce_token_bind = x509
  ```

Use trusts

OpenStack Identity manages authentication and authorization. A trust is an OpenStack Identity extension that enables delegation and, optionally, impersonation through `keystone`. A trust extension defines a relationship between:

Trustor	The user delegating a limited set of their own rights to another user.
Trustee	The user trust is being delegated to, for a limited time.

The trust can eventually allow the trustee to impersonate the trustor. For security reasons, some safeties are added. For example, if a trustor loses a given role, any trusts the user issued with that role, and the related tokens, are automatically revoked.

The delegation parameters are:

User ID

The user IDs for the trustor and trustee.

Privileges

The delegated privileges are a combination of a tenant ID and a number of roles that must be a subset of the roles assigned to the trustor.

If you omit all privileges, nothing is delegated. You cannot delegate everything.

Delegation depth

Defines whether or not the delegation is recursive. If it is recursive, defines the delegation chain length.

Specify one of the following values:

- `0`. The delegate cannot delegate these permissions further.

	• `1`. The delegate can delegate the permissions to any set of delegates but the latter cannot delegate further. • `inf`. The delegation is infinitely recursive.
Endpoints	A list of endpoints associated with the delegation. This parameter further restricts the delegation to the specified endpoints only. If you omit the endpoints, the delegation is useless. A special value of `all_endpoints` allows the trust to be used by all endpoints associated with the delegated tenant.
Duration	(Optional) Comprised of the start time and end time for the trust.

Caching layer

OpenStack Identity supports a caching layer that is above the configurable subsystems (for example, token, assignment). OpenStack Identity uses the `dogpile.cache` library which allows flexible cache back ends. The majority of the caching configuration options are set in the `[cache]` section of the `keystone.conf` file. However, each section that has the capability to be cached usually has a caching boolean value that toggles caching.

So to enable only the token back end caching, set the values as follows:

```
[cache]
enabled=true

[assignment]
caching=false

[token]
caching=true
```

Note

For the Juno release, the default setting is enabled for subsystem caching, but the global toggle is disabled. As a result, no caching in available unless the global toggle for `[cache]` is enabled by setting the value to `true`.

Caching for tokens and tokens validation

The token system has a separate `cache_time` configuration option, that can be set to a value above or below the global `expiration_time` default, allowing for different caching behavior from the other systems in OpenStack Identity. This option is set in the `[token]` section of the configuration file.

The token revocation list cache time is handled by the configuration option `revocation_cache_time` in the `[token]` section. The revocation list is refreshed whenever a token is revoked. It typically sees significantly more requests than specific token retrievals or token validation calls.

Here is a list of actions that are affected by the cached time: getting a new token, revoking tokens, validating tokens, checking v2 tokens, and checking v3 tokens.

The delete token API calls invalidate the cache for the tokens being acted upon, as well as invalidating the cache for the revoked token list and the validate/check token calls.

Token caching is configurable independently of the `revocation_list` caching. Lifted expiration checks from the token drivers to the token manager. This ensures that cached tokens will still raise a `TokenNotFound` flag when expired.

For cache consistency, all token IDs are transformed into the short token hash at the provider and token driver level. Some methods have access to the full ID (PKI Tokens), and some methods do not. Cache invalidation is inconsistent without token ID normalization.

Caching around assignment CRUD

The assignment system has a separate `cache_time` configuration option, that can be set to a value above or below the global `expiration_time` default, allowing for different caching behavior from the other systems in Identity service. This option is set in the `[assignment]` section of the configuration file.

Currently `assignment` has caching for `project`, `domain`, and `role` specific requests (primarily around the CRUD actions). Caching is currently not implemented on grants. The `list` methods are not subject to caching.

Here is a list of actions that are affected by the assignment: assign domain API, assign project API, and assign role API.

The create, update, and delete actions for domains, projects and roles will perform proper invalidations of the cached methods listed above.

Note

If a read-only `assignment` back end is in use, the cache will not immediately reflect changes on the back end. Any given change may take up to the `cache_time` (if set in the `[assignment]` section of the configuration file) or the global `expiration_time` (set in the `[cache]` section of the configuration file) before it is reflected. If this type of delay (when using a read-only `assignment` back end) is an issue, it is recommended that caching be disabled on `assignment`. To disable caching specifically on `assignment`, in the `[assignment]` section of the configuration set `caching` to `False`.

For more information about the different back ends (and configuration options), see:

- `dogpile.cache.backends.memory`
- `dogpile.cache.backends.memcached`

  ### Note

  The memory back end is not suitable for use in a production environment.

- `dogpile.cache.backends.redis`
- `dogpile.cache.backends.file`

- keystone.common.cache.backends.mongo

Example 2.1. Configure the Memcached back end

The following example shows how to configure the memcached back end:

```
[cache]

enabled = true
backend = dogpile.cache.memcached
backend_argument = url:127.0.0.1:11211
```

You need to specify the URL to reach the `memcached` instance with the `backend_argument` parameter.

User CRUD

Identity provides a user CRUD (Create, Read, Update, and Delete) filter that can be added to the `public_api` pipeline. The 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-paste.ini`. For example:

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

[pipeline:public_api]
pipeline = sizelimit url_normalize request_id build_auth_context token_auth
 admin_token_auth json_body 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 current tokens for the user are invalidated.

Note

Only use a KVS back end for tokens when testing.

Logging

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

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

Start the Identity services

To start the services for Identity, 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 + project_name env variables
export OS_USERNAME=admin
export OS_PASSWORD=secrete
export OS_PROJECT_NAME=admin
openstack user list
openstack project create demo

# Using user + password + project-name flags
openstack --os-username admin --os-password secrete --os-project-name admin user list
openstack --os-username admin --os-password secrete --os-project-name admin project create demo
```

Authentication middleware with user name and password

You can also configure Identity authentication middleware using the `admin_user` and `admin_password` options.

Note

The `admin_token` option is deprecated, and no longer used for configuring auth_token middleware.

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`.

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

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

[keystone_authtoken]
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
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.

Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

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

```
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:filter_factory
auth_uri = http://controller:5000/v2.0
identity_uri = http://controller:35357
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.

Note

Comment out any `auth_host`, `auth_port`, and `auth_protocol` options because the `identity_uri` option replaces them.

Identity API protection with role-based access control (RBAC)

Like most OpenStack projects, Identity supports the protection of its APIs by defining policy rules based on an RBAC approach. Identity stores a reference to a policy JSON file in the main Identity configuration file, `keystone.conf`. Typically this file is named `policy.json`, and it contains the rules for which roles have access to certain actions in defined services.

Each Identity API v3 call has a line in the policy file that dictates which level of governance of access applies.

```
API_NAME: RULE_STATEMENT or MATCH_STATEMENT
```

Where:

RULE_STATEMENT can contain *RULE_STATEMENT* or *MATCH_STATEMENT*.

MATCH_STATEMENT is a set of identifiers that must match between the token provided by the caller of the API and the parameters or target entities of the API call in question. For example:

```
"identity:create_user": [["role:admin", "domain_id:%(user.domain_id)s"]]
```

Indicates that to create a user, you must have the admin role in your token and the `domain_id` in your token (which implies this must be a domain-scoped token) must match the `domain_id` in the user object that you are trying to create. In other words, you must have the admin role on the domain in which you are creating the user, and the token that you use must be scoped to that domain.

Each component of a match statement uses this format:

```
ATTRIB_FROM_TOKEN:CONSTANT or ATTRIB_RELATED_TO_API_CALL
```

The Identity service expects these attributes:

Attributes from token: `user_id`, the `domain_id` or `project_id` depending on the scope, and the list of roles you have within that scope.

Attributes related to API call: Any parameters passed into the API call are available, along with any filters specified in the query string. You reference attributes of objects passed with an object.attribute syntax (such as, `user.domain_id`). The target objects of an API are also available using a target.object.attribute syntax. For instance:

```
"identity:delete_user": [["role:admin", "domain_id:%(target.user.
domain_id)s"]]
```

would ensure that Identity only deletes the user object in the same domain as the provided token.

Every target object has an `id` and a `name` available as `target.*OBJECT*.id` and `target.*OBJECT*.name`. Identity retrieves other attributes from the database, and the attributes vary between object types. The Identity service filters out some database fields, such as user passwords.

List of object attributes:

```
role:
     target.role.id
     target.role.name

 user:
     target.user.default_project_id
     target.user.description
     target.user.domain_id
     target.user.enabled
     target.user.id
     target.user.name

 group:
     target.group.description
```

```
    target.group.domain_id
    target.group.id
    target.group.name

domain:
    target.domain.enabled
    target.domain.id
    target.domain.name

project:
    target.project.description
    target.project.domain_id
    target.project.enabled
    target.project.id
    target.project.name
```

The default `policy.json` file supplied provides a somewhat basic example of API protection, and does not assume any particular use of domains. Refer to `policy.v3cloudsample.json` as an example of multi-domain configuration installations where a cloud provider wants to delegate administration of the contents of a domain to a particular admin domain. This example policy file also shows the use of an admin_domain to allow a cloud provider to enable cloud administrators to have wider access across the APIs.

A clean installation could start with the standard policy file, to allow creation of the admin_domain with the first users within it. You could then obtain the domain_id of the admin domain, paste the ID into a modified version of `policy.v3cloudsample.json`, and then enable it as the main policy file.

Troubleshoot the Identity service

To troubleshoot the Identity service, review the logs in the `/var/log/keystone/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 Identity 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://controller:5000/v2.0
identity_uri = http://controller:35357
admin_tenant_name = service
admin_user = glance
```

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 Identity. To troubleshoot, try to talk to Identity to make sure it correctly serves 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 configuration file and restart the Identity 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 Identity. 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 Identity 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, Identity 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 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
```

Flush expired tokens from the token database table

As you generate tokens, the token database table on the Identity server grows. To clear the token table, an administrative user must run the **keystone-manage token_flush** command to flush the tokens. When you flush tokens, expired tokens are deleted and traceability is eliminated.

Use **cron** to schedule this command to run frequently based on your workload. For large workloads, running it every minute is recommended.

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.

Dashboard resources:

- To customize the dashboard, see the section called “Customize the dashboard” [51].
- To set up session storage for the dashboard, see the section called “Set up session storage for the dashboard” [53].
- To deploy the dashboard, see the Horizon documentation.
- To launch instances with the dashboard, see the *OpenStack End User Guide*.

Customize the dashboard

Once you have the dashboard installed you can customize the way it looks and feels to suit your own needs.

Note

The OpenStack dashboard by default on Ubuntu installs the `openstack-dashboard-ubuntu-theme` package.

If you do not want to use this theme you can remove it and its dependencies using the following command:

```
# apt-get remove --auto-remove openstack-dashboard-ubuntu-theme
```

Note

This guide focuses on the `local_settings.py` file, stored in `/openstack-dashboard/openstack_dashboard/local/`.

This guide is adapted from How To Custom Brand The OpenStack "Horizon" Dashboard.

The following can easily be customized:

- Site colors

- Logo
- HTML title
- Site branding link
- Help URL

Logo and site colors

1. Create two logo files, png format, with transparent backgrounds using the following sizes:
   - Login screen: 365 x 50
   - Logged in banner: 216 x 35
2. Upload your new images to the following location: `/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/img/`
3. Create a CSS style sheet in the following directory: `/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/css/`
4. 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;
}
```

5. Open the following HTML template in an editor of your choice: `/usr/share/openstack-dashboard/openstack_dashboard/templates/_stylesheets.html`
6. Add a line to include your newly created style sheet. For example `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' />
    ...
```

7. **Restart Apache:**

   On Ubuntu:

   ```
   # service apache2 restart
   ```

   On Fedora, RHEL, CentOS:

   ```
   # service httpd restart
   ```

   On openSUSE:

   ```
   # service apache2 restart
   ```

8. To view your changes simply reload your dashboard. If necessary go back and modify your CSS file as appropriate.

HTML title

1. Set the HTML title, which appears at the top of the browser window, by adding the following line to `local_settings.py`:

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

2. Restart Apache for this change to take effect.

HTML title

1. The logo also acts as a hyperlink. The default behavior is to redirect to `horizon:user_home`. To change this, add the following attribute to `local_settings.py`

   ```
   SITE_BRANDING_LINK = "http://example.com"
   ```

2. Restart Apache for this change to take effect.

Help URL

1. By default the help URL points to http://docs.openstack.org. Change this by editing the following arritbute to the URL of your choice in `local_settings.py`

   ```
   'help_url': "http://openstack.mycompany.org",
   ```

2. Restart Apache for this change to take effect.

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 = {
  'default' : {
    '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 = {
  'default': {
    '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.contrib.sessions.backends.db'
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. To avoid a warning when you restart Apache on Ubuntu, create a `blackhole` directory in the dashboard directory, as follows:

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

9. **Restart and refresh Apache**

   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 queuing 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 can 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 user name 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.

Block storage

OpenStack provides two classes of the block storage: ephemeral storage and persistent volume.

Ephemeral storage

An ephemeral storage includes a root ephemeral volume and an additional ephemeral volume.

The root disk is associated with an instance, and exists only for the life of this very instance. Generally, it is used to store an instance`s root file system, persists across the guest operating system reboots, and is removed on an instance deletion. The amount of the root ephemeral volume is defined by the flavor of an instance.

In addition to the ephemeral root volume, all default types of flavors, except `m1.tiny`, which is the smallest one, provide an additional ephemeral block device sized between 20 and 160 GB (a configurable value to suit an environment). It is represented as a raw block device with no partition table or file system. A cloud-aware operating system can discover, format, and mount such a storage device. OpenStack Compute defines the default file system for different operating systems as Ext4 for Linux distributions, VFAT for non-Linux and non-Windows operating systems, and NTFS for Windows. However, it is possible to specify any other filesystem type by using `virt_mkfs` or `default_ephemeral_format` configuration options.

Note

For example, the `cloud-init` package included into an Ubuntu's stock cloud image, by default, formats this space as an Ext4 file system and mounts it on

`/mnt`. This is a cloud-init feature, and is not an OpenStack mechanism. OpenStack only provisions the raw storage.

Persistent volume

A persistent volume is represented by a persistent virtualized block device independent of any particular instance, and provided by OpenStack Block Storage.

Only a single configured instance can access a persistent volume. Multiple instances cannot access a persistent volume. This type of configuration requires a traditional network file system to allow multiple instances accessing the persistent volume. It also requires a traditional network file system like NFS, 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.

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

Note

A persistent volume does 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. For more information and configuration options about this compatibility API, see the *OpenStack Configuration Reference*.

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 virtual machine deletion. It is also possible to put an operating system on a persistent volume in the OpenStack Block Storage volume system. This gives a more traditional persistent system that accumulates states which are preserved on the OpenStack Block Storage volume across the deletion and re-creation of the virtual machine. To get a list of available images on your system, run:

```
$ nova image-list
+--------------------------------------+-------------------------------+--------+--------------------------------------+
| ID                                   | Name                          | Status | Server                               |
+--------------------------------------+-------------------------------+--------+--------------------------------------+
| aee1d242-730f-431f-88c1-87630c0f07ba | Ubuntu 14.04 cloudimg amd64   | ACTIVE |                                      |
| 0b27baa1-0ca6-49a7-b3f4-48388e440245 | Ubuntu 14.10 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
 | Is_Public |
+-----+-----------+-----------+------+-----------+------+-------+-------------
+-----------+
| 1   | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0
 | True      |
| 2   | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
 | True      |
| 3   | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
 | True      |
| 4   | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
 | True       |
| 5   | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
 | True       |
+-----+-----------+-----------+------+-----------+------+-------+------------
+-----------+
```

Compute service architecture

These basic categories describe the service architecture and information about the cloud controller.

API server

At the heart of the cloud framework is an API server, which 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 handled 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 they are 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 executing it. 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 during 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

Images and instances

Disk images provide templates for virtual machine file systems. The Image service controls 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 service 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, the amount of RAM available to it, and the size of its ephemeral disks. Users must select from the set of available flavors defined on their cloud. OpenStack provides a number of predefined flavors that you can edit or add to.

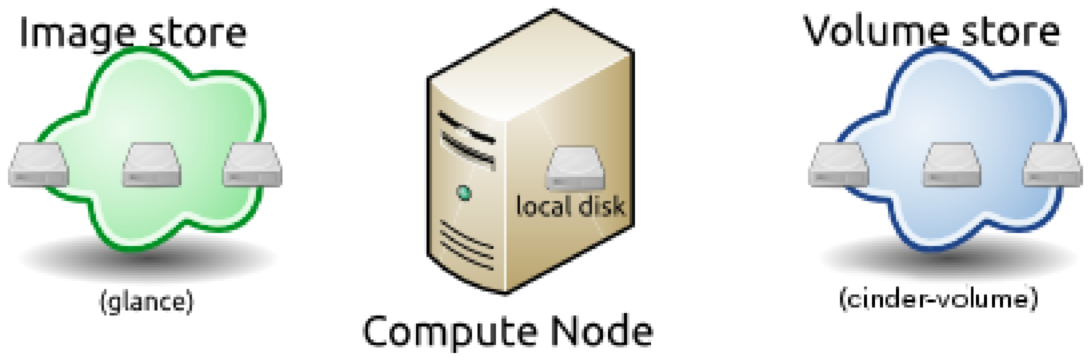
Note

- 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” [90] or Flavors 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.

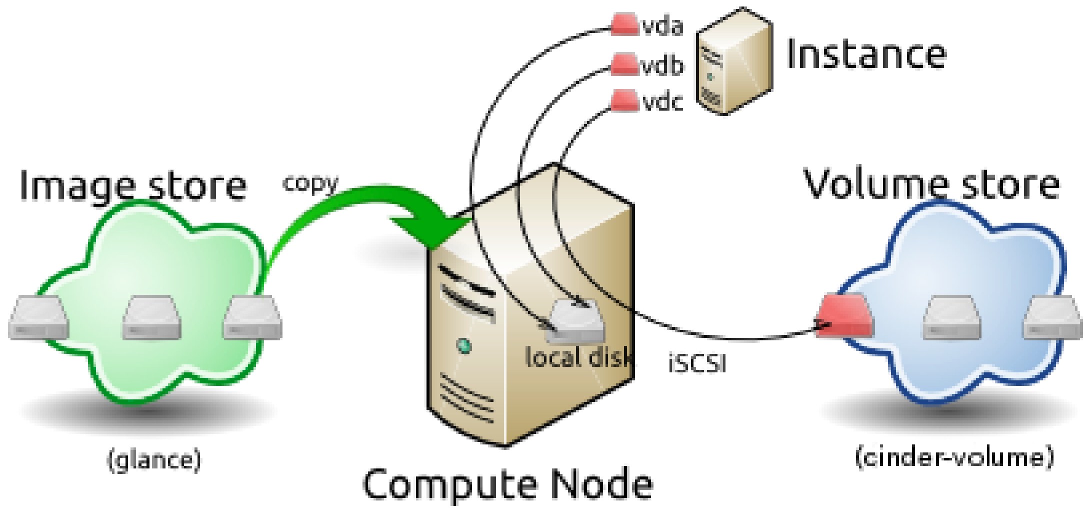
This diagram shows the system state prior to launching an instance. The image store, fronted by the Image service (glance) has a number of predefined images. Inside the cloud, a compute node contains the available vCPU, memory, and local disk resources. Additionally, the `cinder-volume` service provides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

Figure 4.1. Base image state with no running instances



To launch an instance select an image, flavor, and any 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, labeled `vda` in this diagram. Your instances will start up faster if you use smaller images, as less data needs to be copied across the network.

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

The compute node connects to the attached `cinder-volume` using ISCSI. The `cinder-volume` is mapped to the third disk, labeled `vdc` in this diagram. After the compute node provisions the vCPU and memory resources, the instance boots up from root volume `vda`. The instance runs, and changes data on the disks (highlighted in red on the diagram).

If the volume store is located on a separate network, the `my_block_storage_ip` option specified in the storage node configuration file directs image traffic to the compute node.

Note

Some 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 this process.

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

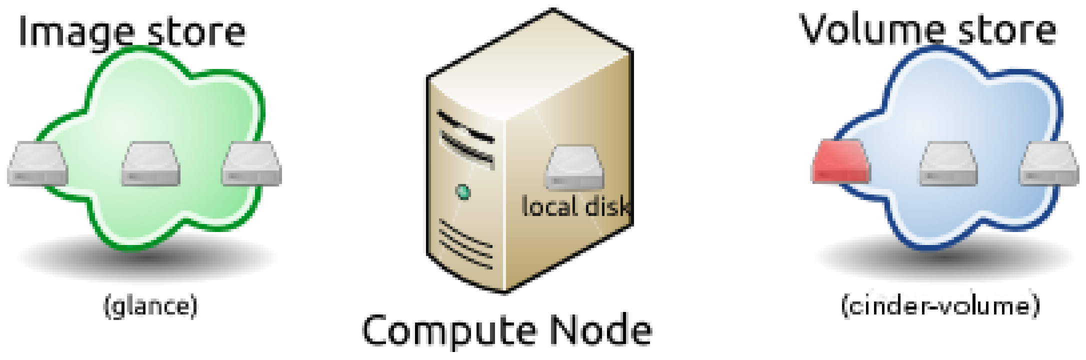


Image management

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 using the 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	The OpenStack highly available service for storing objects.
Block Storage	The OpenStack highly available service for storing blocks.
VMware	ESX/ESXi or vCenter Server target system.
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.
Sheepdog	A distributed storage system for QEMU/KVM.
GridFS	Stores images using MongoDB.

Image properties and property protection

An image property is a key and value pair that the cloud administrator or the image owner attaches to an OpenStack Image Service image, as follows:

- The cloud administrator defines *core* properties, such as the image name.
- The cloud administrator and the image owner can define *additional* properties, such as licensing and billing information.

The cloud administrator can configure any property as *protected*, which limits which policies or user roles can perform CRUD operations on that property. Protected properties are generally additional properties to which only cloud administrators have access.

For unprotected image properties, the cloud administrator can manage core properties and the image owner can manage additional properties.

To configure property protection

To configure property protection, the cloud administrator completes these steps:

1. Define roles or policies in the `policy.json` file:

```
{
    "context_is_admin":  "role:admin",
    "default": "",

    "add_image": "",
    "delete_image": "",
    "get_image": "",
    "get_images": "",
    "modify_image": "",
    "publicize_image": "role:admin",
    "copy_from": "",

    "download_image": "",
    "upload_image": "",

    "delete_image_location": "",
    "get_image_location": "",
    "set_image_location": "",

    "add_member": "",
    "delete_member": "",
    "get_member": "",
    "get_members": "",
    "modify_member": "",

    "manage_image_cache": "role:admin",

    "get_task": "",
    "get_tasks": "",
    "add_task": "",
    "modify_task": "",

    "deactivate": "",
    "reactivate": "",

    "get_metadef_namespace": "",
    "get_metadef_namespaces":"",
    "modify_metadef_namespace":"",
    "add_metadef_namespace":"",

    "get_metadef_object":"",
    "get_metadef_objects":"",
    "modify_metadef_object":"",
    "add_metadef_object":"",

    "list_metadef_resource_types":"",
    "get_metadef_resource_type":"",
    "add_metadef_resource_type_association":"",

    "get_metadef_property":"",
    "get_metadef_properties":"",
    "modify_metadef_property":"",
    "add_metadef_property":"",

    "get_metadef_tag":"",
    "get_metadef_tags":"",
    "modify_metadef_tag":"",
    "add_metadef_tag":"",
    "add_metadef_tags":""

}
```

For each parameter, use `"rule:restricted"` to restrict access to all users or `"role:admin"` to limit access to administrator roles. For example:

```
"download_image": "rule:restricted"
"upload_image": "role:admin"
```

2. Define which roles or policies can manage which properties in a property protections configuration file. For example:

```
[x_none_read]
create = context_is_admin
```

```
read = !
update = !
delete = !

[x_none_update]
create = context_is_admin
read = context_is_admin
update = !
delete = context_is_admin

[x_none_delete]
create = context_is_admin
read = context_is_admin
update = context_is_admin
delete = !
```

- A value of @ allows the corresponding operation for a property.
- A value of `!` disallows the corresponding operation for a property.

3. In the `glance-api.conf` file, define the location of a property protections configuration file:

   ```
   property_protection_file = {file_name}
   ```

   This file contains the rules for property protections and the roles and policies associated with it.

   By default, property protections are not enforced.

   If you specify a file name value and the file is not found, the `glance-api` service does not start.

   To view a sample configuration file, see glance-api.conf.

4. Optionally, in the `glance-api.conf` file, specify whether roles or policies are used in the property protections configuration file:

   ```
   property_protection_rule_format = roles
   ```

   The default is `roles`.

   To view a sample configuration file, see glance-api.conf.

Image download: how it works

Prior to starting a virtual machine, the virtual machine image used must be transferred to the compute node from the Image Service. How this works can change depending on the settings chosen for the compute node and the Image service.

Typically, the Compute service will use the image identifier passed to it by the scheduler service and request the image from the Image API. Though images are not stored in glance—rather in a back end, which could be Object Storage, a filesystem or any other supported method—the connection is made from the compute node to the Image service and the image is transferred over this connection. The Image service streams the image from the back end to the compute node.

It is possible to set up the Object Storage node on a separate network, and still allow image traffic to flow between the Compute and Object Storage nodes. Configure the `my_block_storage_ip` option in the storage node configuration to allow block storage traffic to reach the Compute node.

Certain back ends support a more direct method, where on request the Image service will return a URL that can be used to download the image directly from the back-end store. Currently the only store to support the direct download approach is the filesystem store. It can be configured using the `filesystems` option in the `image_file_url` section of the `nova.conf` file on compute nodes.

Compute nodes also implement caching of images, meaning that if an image has been used before it won't necessarily be downloaded every time. Information on the configuration options for caching on compute nodes can be found in the *Configuration Reference*.

Instance building blocks

In OpenStack, the base operating system is usually copied from an image stored in the OpenStack Image service. This 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 that 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 14.04 cloudimg amd64   | ACTIVE |                                      |
| 0b27baa1-0ca6-49a7-b3f4-48388e440245 | Ubuntu 14.10 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 the 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 predefined flavors.

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
 | Is_Public |
+-----+-----------+-----------+------+-----------+------+-------+-------------
+-----------+
| 1   | m1.tiny   | 512       | 1    | 0         |      | 1     | 1.0
 | True      |
| 2   | m1.small  | 2048      | 20   | 0         |      | 1     | 1.0
 | True       |
| 3   | m1.medium | 4096      | 40   | 0         |      | 2     | 1.0
 | True       |
| 4   | m1.large  | 8192      | 80   | 0         |      | 4     | 1.0
 | True       |
| 5   | m1.xlarge | 16384     | 160  | 0         |      | 8     | 1.0
 | True       |
+-----+-----------+-----------+------+-----------+------+-------+-------------
+-----------+
```

By default, administrative users can configure the flavors. You can change this behavior by redefining the access controls for `compute_extension:flavormanage` in `/etc/nova/policy.json` on the `compute-api` server.

Instance management tools

OpenStack provides command-line, web interface, and API-based instance management tools. Third-party management tools are also 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 command-line tools, see the *OpenStack End User Guide*.

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.

Administrative users can specify which compute node their instances run on. To do this, specify the `--availability-zone AVAILABILITY_ZONE:COMPUTE_HOST` parameter.

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` or use the OpenStack Networking service (neutron). This section contains a brief overview of `nova-network`. For more information about OpenStack Networking, see Chapter 7, “Networking” [203].

Networking concepts

Compute assigns a private IP address to each VM instance. Compute makes a distinction between fixed IPs and *floating IP*. Fixed IPs are IP addresses that are assigned to an instance on creation and stay the same until the instance is explicitly terminated. 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.

Note

Currently, Compute with `nova-network` only supports Linux bridge networking that allows virtual interfaces to connect to the outside network through the physical interface.

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 this 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, which 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 this 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 configures instances depending on single-/multi-host mode, alongside each `nova-network`. In this mode, Compute does a bit more configuration. 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 ID and IP address 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 with the **dhcpdiscover** command. These IPs are not assigned to any of the host's network interfaces, only to the guest-side interface for the VM.

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 the network has been configured with all public IPs, or if a hardware router is used (which is a high availability option). In this case, 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 or ebtables entries created for each project and instance to protect against MAC ID or IP 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

This is the default mode for OpenStack Compute. In this mode, Compute creates a VLAN and bridge for each tenant. For multiple-machine installations, 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 the libvirt network filtering facilities. The driver is not tied to any particular network manager; all network managers use the same driver. The driver usually initializes 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 internet flows through `nova-network`. Each mode has benefits and drawbacks. For more on this, see the *Network Topology* section in the *OpenStack Operations Guide*.

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 network managers can also create network bridges such as `br100`.

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 only as a fabric where the actual endpoints are VMs and dnsmasq. Additionally, the internal network interface must be in `promiscuous` mode, so that it can receive packets whose target MAC address is the guest VM, not the host.

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

For flat and flat DHCP modes, use the **nova network-create** command to create a network:

```
$ nova network-create vmnet \
--fixed-range-v4 10.0.0.0/16 --fixed-cidr 10.0.20.0/24 --bridge br100
```

This example uses the following parameters:

- *`--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 using either Flat DHCP Network Manager or VLAN Network Manager. For Compute to operate in IPv4/IPv6 dual-stack mode, use at least `dnsmasq` v2.63. The `nova-network` service is responsible for starting `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:

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

For more information about creating a `dnsmasq` configuration file, see the *OpenStack Configuration Reference*, and the dnsmasq documentation.

`dnsmasq` also acts as a caching DNS server for instances. You can specify the DNS server that `dnsmasq` uses by setting the `dns_server` configuration option in `/etc/nova/nova.conf`. This example configures `dnsmasq` to use Google's public DNS server:

```
dns_server=8.8.8.8
```

`dnsmasq` logs to `syslog` (typically `/var/log/syslog` or `/var/log/messages`, depending on Linux distribution). Logs can be useful for troubleshooting, especially in a situation where VM instances boot successfully but are not reachable over the network.

Administrators can specify the starting point IP address to reserve with the DHCP server (in the format *n.n.n.n*) with this command:

```
$nova-manage fixed reserve --address IP_ADDRESS
```

This reservation only affects which IP address the VMs start at, not the fixed IP addresses that `nova-network` places on the bridges.

Configure Compute to use IPv6 addresses

If you are using OpenStack Compute with `nova-network`, you can put Compute into dual-stack mode, so that it uses both IPv4 and IPv6 addresses for communication. In 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` networking mode, each project uses a different 64-bit global routing prefix. In `FlatDHCPManager` mode, all instances use one 64-bit global routing prefix.

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

Switch into IPv4/IPv6 dual-stack mode

1. For every node running a `nova-*` service, install `python-netaddr`:

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

2. For every node running `nova-network`, 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. On all nodes, edit the `nova.conf` file and 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 the 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 4.1. Description of IPv6 configuration options

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

Compute uses a metadata service for 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. Both APIs are 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`:

```
$ curl http://169.254.169.254/openstack
2012-08-10
2013-04-04
2013-10-17
latest
```

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

```
$ 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.

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`:

```
$ 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"
    },
    "project_id": "f7ac731cc11f40efbc03a9f9e1d1d21f",
    "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`:

```
$ curl http://169.254.169.254/openstack/2012-08-10/user_data
#!/bin/bash
```

```
echo 'Extra user data here'
```

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

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

```
$ 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`:

```
$ 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`:

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

The metadata service is implemented by either the `nova-api` service or the `nova-api-metadata` service. Note that the `nova-api-metadata` service is generally only used when running in multi-host mode, as it retrieves instance-specific metadata. If you are running the `nova-api` service, you must have `metadata` as one of the elements listed in 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` service. In multi-host mode, you can set `metadata_host` to `127.0.0.1`.

For instances to reach the metadata service, the `nova-network` service must configure iptables to NAT port `80` of the `169.254.169.254` address to the IP address specified in `metadata_host` (this defaults to `$my_ip`, which is the IP address of the `nova-network` service) and port specified in `metadata_port` (which defaults to `8775`) in `/etc/nova/nova.conf`.

Note

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

The default Compute service settings assume that `nova-network` and `nova-api` are running on the same host. If this is not the case, in the `/etc/nova/nova.conf` file on the host running `nova-network`, set the `metadata_host` configuration option to the IP address of the host where `nova-api` is running.

Table 4.2. Description of metadata configuration options

Configuration option = Default value	Description
[DEFAULT]	
`metadata_cache_expiration` = *15*	(IntOpt) Time in seconds to cache metadata; 0 to disable metadata caching entirely (not recommended). Increasingthis should improve response times of the metadata API when under heavy load. Higher values may increase memoryusage and result in longer times for host metadata changes to take effect.
`metadata_host` = *$my_ip*	(StrOpt) The IP address for the metadata API server
`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

You need to enable **ping** and **ssh** on your VMs for network access. This can be done with either the **nova** or **euca2ools** commands.

Note

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

Enable ping and SSH with **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
```

Enable ping and SSH with **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 have run these commands and still cannot ping or SSH your instances, check the number of running `dnsmasq` processes, there should be two. If not, kill the processes and restart the service with these commands: command:

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

Configure public (floating) IP addresses

This section describes how to configure floating IP addresses with `nova-network`. For information about doing this with OpenStack Networking, see the section called “L3 routing and NAT” [242].

Private and public IP addresses

In this section, the term *floating IP address* is used to refer to an IP address, usually public, that you can dynamically add to a running virtual instance.

Every virtual instance is automatically assigned a private IP address. You can choose to assign a public (or floating) IP address instead. OpenStack Compute uses network address translation (NAT) to assign floating IPs to virtual instances.

To be able to assign a floating IP address, edit the `/etc/nova/nova.conf` file to specify which interface the `nova-network` service should bind public IP addresses to:

```
public_interface=VLAN100
```

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

Traffic between VMs using floating IPs

Floating IPs are implemented by using a source NAT (SNAT rule in iptables), so security groups can sometimes 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 setup. 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. This configuration is also required if the VMs in the source group have floating IPs.

Enable IP forwarding

IP forwarding is disabled by default on most Linux distributions. You will need to enable it in order to use floating IPs.

Note

IP forwarding only needs to be enabled on the nodes that run `nova-network`. However, you will need to enable it on all compute nodes if you use `multi_host` mode.

To check if IP forwarding is enabled, run:

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

Alternatively, run:

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

In these examples, IP forwarding is disabled.

To enable IP forwarding dynamically, run:

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

Alternatively, run:

```
# 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 this command to apply the changes:

```
# sysctl -p
```

You can also apply the changes by restarting the network service:

- on Ubuntu, Debian:

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

- on RHEL, Fedora, CentOS, openSUSE and SLES:

```
# service network restart
```

Create a list of available floating IP addresses

Compute maintains a list of floating IP addresses that are available for assigning to instances. Use the **nova-manage floating create** command to add entries to the list:

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

Use these **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 more 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 `nova-network` to automatically allocate and assign a floating IP address to virtual instances when they are launched. Add this line to the `/etc/nova/nova.conf` file:

```
auto_assign_floating_ip=True
```

Save the file, and restart `nova-network`

Note

If this option is enabled, but all floating IP addresses have already been allocated, the **nova boot** command will fail.

Remove a network from a project

You cannot delete a network that has been associated to a project. This section describes the procedure for dissociating it so that it can be deleted.

In order to disassociate the network, you will need the ID of the project it has been associated to. To get the project ID, you will need to be an administrator.

Disassociate the network from the project using the **scrub** command, with the project ID as the final parameter:

```
# nova-manage project scrub --project ID
```

Multiple interfaces for instances (multinic)

The multinic feature allows you to use more than one interface with your instances. This is useful in several scenarios:

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

Each VIP represents a separate network with its own IP block. Every network mode has its own set of changes regarding multinic usage:

Figure 4.4. multinic flat manager

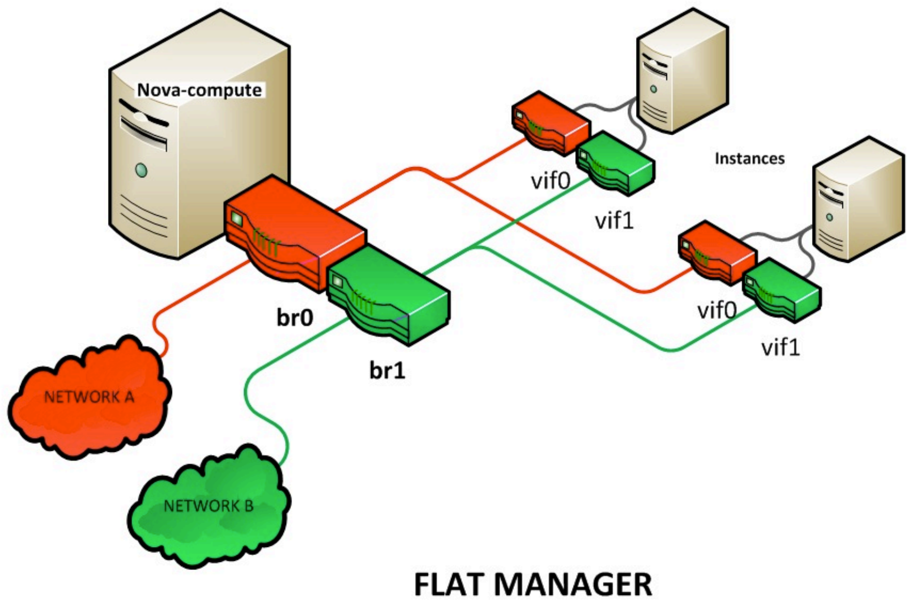


Figure 4.5. multinic flatdhcp manager

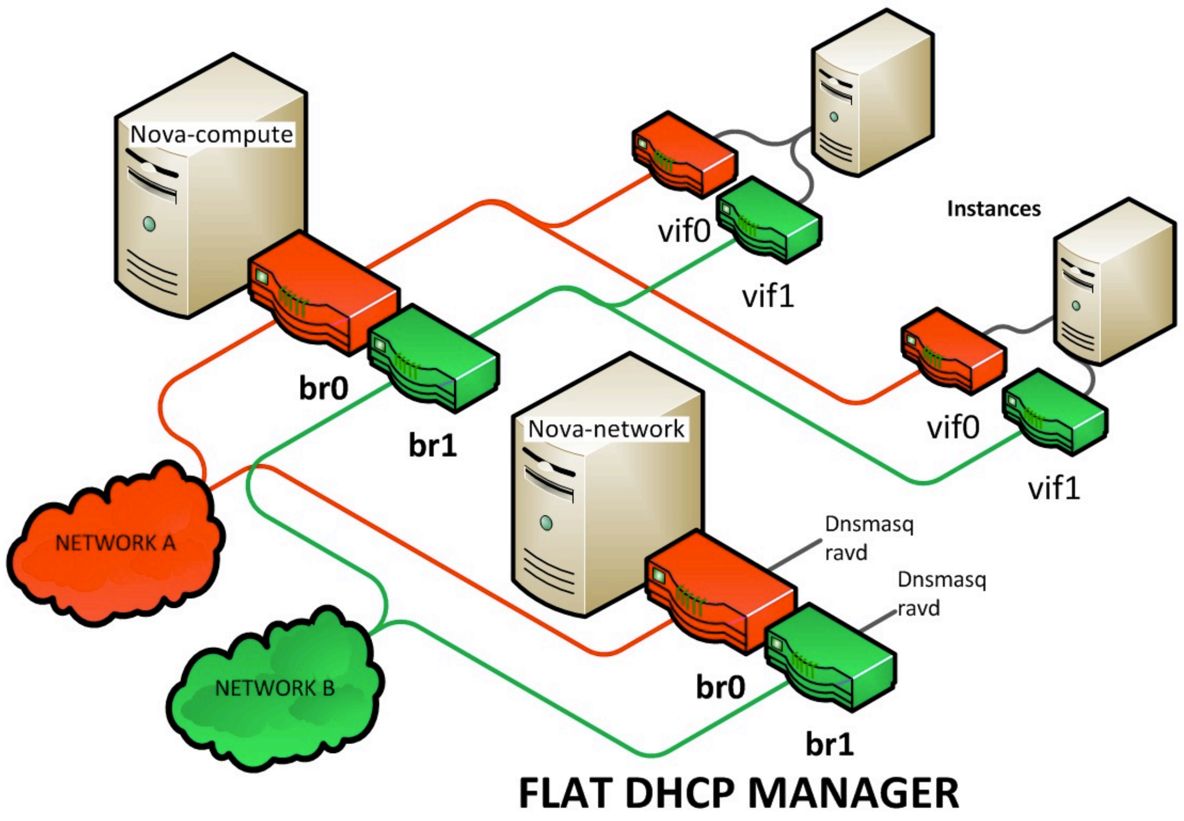
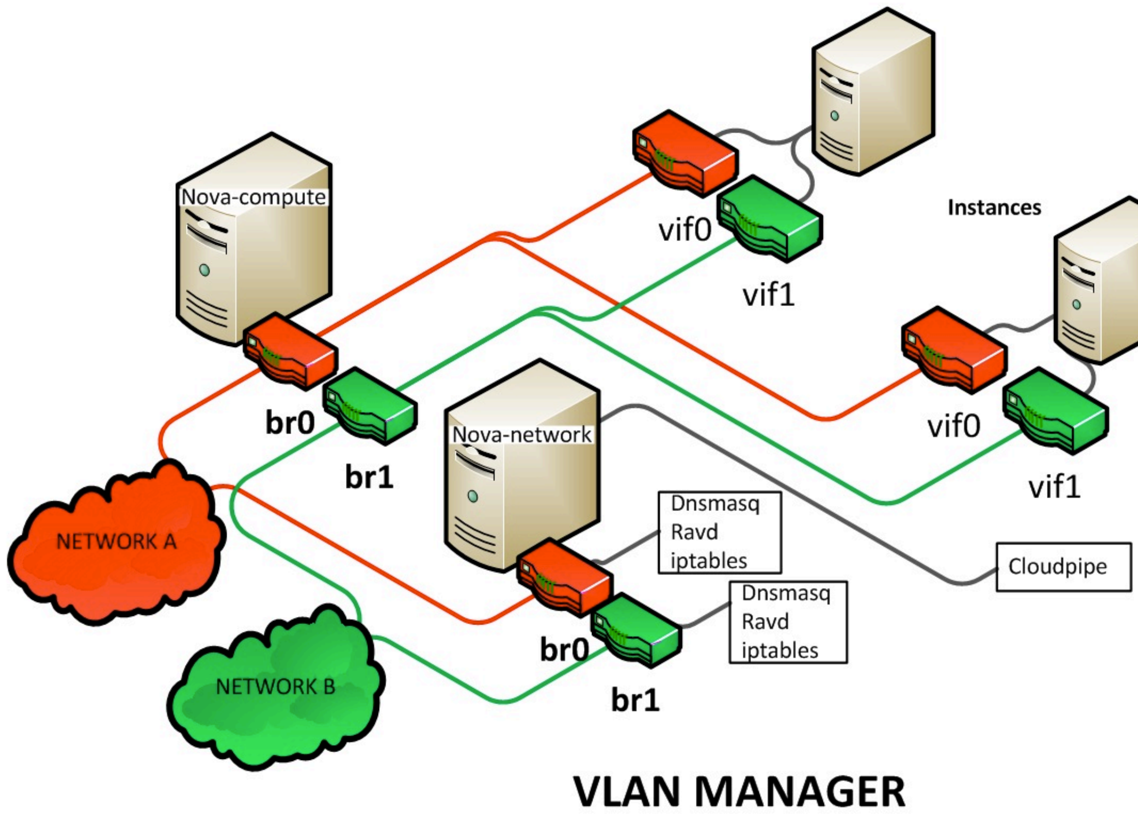


Figure 4.6. multinic VLAN manager



Using multinic

In order to use multinic, create two networks, and attach them to the tenant (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
```

Each new instance will now receive two IP addresses from their 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 you start the second interface on the instance, or it won't be reachable through the second IP.

This example demonstrates how to set up the interfaces within the instance. This is the configuration that needs to be applied inside the image.

Edit the `/etc/network/interfaces` file:

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

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

If the Virtual Network Service Neutron is installed, you can specify the networks to attach to the interfaces by using the `--nic` flag with the **nova** command:

```
$ nova boot --image ed8b2a37-5535-4a5f-a615-443513036d71 --flavor 1 --nic net-id=NETWORK1_ID --nic net-id=NETWORK2_ID test-vm1
```

Troubleshooting Networking

Cannot reach floating IPs

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

- Check that 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 |              |
  +-------------+-----------+---------+-----------+--------------+
  ```

- Check the NAT rules have been added to `iptables` on the node that is running `nova-network`:

  ```
  # 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 (`68.99.26.170` in this example), has been added to your public interface. You should see the address in the listing when you use the **ip addr** command:

  ```
  $ 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

You cannot **ssh** to an instance with a public IP from within the same server because the routing configuration does not allow it.

- 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, run:

  ```
  # sysctl -w net.ipv4.conf.ETH2.rp_filter=0
  ```

  If this solves the problem, add the following line to /etc/sysctl.conf so that the reverse-path filter is persistent:

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

Temporarily disable the firewall

To help debug networking issues with reaching VMs, you can disable the firewall by setting this 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 the network to your instance is slow, or if you find that running certain operations are slower than they should be (for example, **sudo**), packet loss could be 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 nova-network.

One way to check whether this is the problem is to open 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 to 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 correct, the command hangs for several seconds before returning.

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 disabling filtering, and running these 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 the changes are persistent:

   ```
   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

With KVM hypervisors, instances running Ubuntu 12.04 sometimes lose network connectivity after functioning properly for a period of time. Try 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:

```
# modprobe vhost_net
```

Note

Loading the module has no effect on running instances.

System administration

To effectively administer Compute, you must understand how the different installed nodes interact with each other. Compute can be installed in many different ways using multiple servers, but generally multiple compute nodes control the virtual servers and a cloud controller node contains the remaining Compute services.

The Compute cloud works using a series of daemon processes named `nova-*` that exist persistently on the host machine. 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. A WSGI app routes and authenticates requests. Supports the EC2 and OpenStack APIs. A `nova.conf` configuration file is created when Compute is installed.

- `nova-cert`: manages certificates.
- `nova-compute`: manages virtual machines. Loads a Service object, and exposes the public methods on ComputeManager through a Remote Procedure Call (RPC).
- `nova-conductor`: provides database-access support for Compute nodes (thereby reducing security risks).
- `nova-consoleauth`: manages console authentication.
- `nova-objectstore`: a simple file-based storage system for images that replicates most of the S3 API. It can be replaced with OpenStack Image service and either a simple image manager or OpenStack Object Storage as the virtual machine image storage facility. It must exist on the same node as `nova-compute`.
- `nova-network`: manages floating and fixed IPs, DHCP, bridging and VLANs. Loads a Service object which exposes the public methods on one of the subclasses of `NetworkManager`. Different networking strategies are available by changing the `network_manager` configuration option to `FlatManager`, `FlatDHCPManager`, or `VLANManager` (defaults to `VLANManager` if nothing is specified).
- `nova-scheduler`: dispatches requests for new virtual machines to the correct node.
- `nova-novncproxy`: provides a VNC proxy for browsers, allowing VNC consoles to access virtual machines.

Note

Some services have drivers that change how the service implements its core functionality. For example, the `nova-compute` service supports drivers that let you choose which hypervisor type it can use. `nova-network` and `nova-scheduler` also have drivers.

Manage Compute users

Access to the Euca2ools (ec2) API is controlled by an access key and a 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 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 |             |
+--------------------------------------+-----------+--------------+------
+-------------+-------------+
```

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 4.3. 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.

Element	Description
`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.
`Ephemer-al`	Specifies the size of a secondary ephemeral data disk. This is an empty, un-formatted 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. This parameter is only available for Xen or NSX based systems.
`Is_Public`	Boolean value, whether flavor is available to all users or private to the ten-ant 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 be-havior, 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
```

Use these optional parameters to control 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 config-ured 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 con-sume more than the quota worth of runtime. The val-ue should be in range `[1000, 1000000]`. A period with value 0 means no value.

- `cpu_limit`. Specifies the upper limit for VMware machine CPU allocation in MHz. This parameter ensures that a machine never uses more than the defined amount of CPU time. It can be used to enforce a limit on the machine's CPU performance.

- `cpu_reservation`. Specifies the guaranteed minimum CPU reservation in MHz for VMware. This means that if needed, the machine will definitely get allocated the reserved amount of CPU cycles.

- `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
 quota: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

Bandwidth I/O

The vif I/O options are:

- vif_inbound_ average
- vif_inbound_burst
- vif_inbound_peak

- vif_outbound_ average
- vif_outbound_burst
- vif_outbound_peak

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 child elements 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 the maximum rate at which a 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.

Below example sets network traffic bandwidth limits for existing flavor as follow:

- Outbound traffic:
  - average: 256 Mbps (32768 kilobytes/second)
  - peak: 512 Mbps (65536 kilobytes/second)
  - burst: 100 ms
- Inbound traffic:
  - average: 256 Mbps (32768 kilobytes/second)
  - peak: 512 Mbps (65536 kilobytes/second)
  - burst: 100 ms

```
$ nova flavor-key nlimit set
 quota:vif_outbound_average=32768
$ nova flavor-key nlimit set
 quota:vif_outbound_peak=65536
$ nova flavor-key nlimit set
 quota:vif_outbound_burst=6553
$ nova flavor-key nlimit set
 quota:vif_inbound_average=16384
$ nova flavor-key nlimit set
 quota:vif_inbound_peak=32768
$ nova flavor-key nlimit set
 quota:vif_inbound_burst=3276
```

Note

All the speed limit values in above example are specified in kilobytes/second. And burst values are in kilobytes.

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.

Project private flavors

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
```

Compute service node firewall requirements

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

This procedure modifies the `iptables` firewall to allow incoming connections to the Compute services.

Configuring the service-node firewall

1. Log in to the server that hosts the Compute service, as `root`.
2. Edit the `/etc/sysconfig/iptables` file, to add an INPUT rule that allows TCP traffic on ports from `5900` to `5999`. Make sure the new rule appears before any INPUT rules that REJECT traffic:

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

3. Save the changes to `/etc/sysconfig/iptables`, and restart the `iptables` service to pick up the changes:

   ```
   $ service iptables restart
   ```

4. Repeat this process for each Compute service node.

Injecting the administrator password

Compute can generate a random administrator (root) password and inject that password into an instance. If this feature is enabled, users 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.

Password injection using the dashboard

By default, the dashboard will display the `admin` password and allow the user to modify it.

If you do not want to support password injection, disable the password fields by editing the dashboard's `local_settings` file. On Fedora/RHEL/CentOS, the file location is `/etc/openstack-dashboard/local_settings`. On Ubuntu and Debian, it is `/etc/openstack-dashboard/local_settings.py`. On openSUSE and SUSE Linux Enterprise Server, it is `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`

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

Password injection on libvirt-based hypervisors

For hypervisors that use the libvirt back end (such as KVM, QEMU, and LXC), admin password injection is disabled by default. To enable it, set this option in `/etc/nova/nova.conf`:

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

When enabled, Compute will modify the password of the admin account by editing the `/etc/shadow` file inside 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, and it has been configured to allow users to **ssh** as the root user. This is not the case for Ubuntu cloud images which, by default, do not allow users to **ssh** to the root account.

Password injection and XenAPI (XenServer/XCP)

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

Password injection and Windows images (all hypervisors)

For Windows virtual machines, configure the Windows image to retrieve the admin password on boot by installing an agent such as cloudbase-init.

Manage the cloud

System administrators can use **nova** client and **Euca2ools** commands to manage their clouds.

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.

Managing the cloud with nova client

1. The python-novaclient package provides a `nova` shell that enables Compute API interactions from the command line. Install the client, and provide your user name and password (which can be set as environment variables for convenience), for the ability to administer the cloud from the command line.

   To install python-novaclient, download the tarball from http://pypi.python.org/pypi/python-novaclient/#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, run:

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

2. Confirm the installation was successful:

```
$ 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_VERSION]
            [--os-cacert CA_CERTIFICATE] [--insecure]
            [--bypass-url BYPASS_URL]
            SUBCOMMAND ...
```

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

```
$ nova help SUBCOMMAND
```

For a complete list of **nova** commands and parameters, see the *OpenStack Command-Line Reference*.

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. The Identity Service will give you an authentication endpoint, which Compute recognizes as OS_AUTH_URL.

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

Managing the cloud with euca2ools

The **euca2ools** command-line tool provides a command line interface to EC2 API calls. For more information about **euca2ools**, 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. 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       |
  +----------------------------------+-----------+--------------+-----------
  +---------------+
  ```

Logging

Logging module

Logging behavior can be changed by creating a configuration file. To specify the configuration file, add this line to the /etc/nova/nova.conf file:

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

To change the logging level, add DEBUG, INFO, WARNING, or ERROR as a parameter.

The logging configuration file is an INI-style configuration file, which must contain a section called logger_nova. This 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 is less verbose than the default DEBUG setting).

For more about the logging configuration syntax, including the handlers and quaname variables, see the Python documentation on logging configuration files.

For an example logging.conf file with various defined handlers, see the *OpenStack Configuration Reference*.

Syslog

OpenStack Compute services can send logging information to syslog. This is useful if you want to use rsyslog to forward logs to a remote machine. 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. Open these configuration files:

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

In each configuration file, add these lines:

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

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

Note

Although this example 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 can capture logging information at different severity levels for different services. `syslog` allows you to define up to eight local facilities, `LOCAL0, LOCAL1, ..., LOCAL7`. For more information, see the `syslog` documentation.

Rsyslog

`rsyslog` is useful for setting up a centralized log server across multiple machines. This section briefly describe the configuration to set up an `rsyslog` server. A full treatment of `rsyslog` is beyond the scope of this book. This section assumes `rsyslog` has already been installed on your hosts (it is installed by default on 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. This example uses *`COMPUTE_01`* as the 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 the file, restart the `rsyslog` service. Error-level log messages on the compute hosts should now be sent to the log server.

Serial console

The serial console provides a way to examine kernel output and other system messages during troubleshooting if the instance lacks network connectivity.

OpenStack Icehouse and earlier supports read-only access using the serial console using the **os-GetSerialOutput** server action. Most cloud images enable this feature by default. For more information, see Troubleshoot Compute.

OpenStack Juno and later supports read-write access using the serial console using the **os-GetSerialConsole** server action. This feature also requires a websocket client to access the serial console.

Configuring read-write serial console access

On a compute node, edit the `/etc/nova/nova.conf` file:

1. In the `[serial_console]` section, enable the serial console:

   ```
   [serial_console]
   ...
   enabled = true
   ```

2. In the `[serial_console]` section, configure the serial console proxy similar to graphical console proxies:

   ```
   [serial_console]
   ...
   base_url = ws://controller:6083/
   listen = 0.0.0.0
   proxyclient_address = MANAGEMENT_INTERFACE_IP_ADDRESS
   ```

   The `base_url` option specifies the base URL that clients receive from the API upon requesting a serial console. Typically, this refers to the host name of the controller node.

   The `listen` option specifies the network interface `nova-compute` should listen on for virtual console connections. Typically, 0.0.0.0 will enable listening on all interfaces.

   The `proxyclient_address` option specifies which network interface the proxy should connect to. Typically, this refers to the IP address of the management interface.

When you enable read-write serial console access, Compute will add serial console information to the Libvirt XML file for the instance. For example:

```
<console type='tcp'>
  <source mode='bind' host='127.0.0.1' service='10000'/>
  <protocol type='raw'/>
  <target type='serial' port='0'/>
  <alias name='serial0'/>
</console>
```

Accessing the serial console on an instance

1. Use the **nova get-serial-proxy** command to retrieve the websocket URL for the serial console on the instance:

   ```
   $ nova get-serial-proxy INSTANCE_NAME
   +--------
   +-----------------------------------------------------------------+
   | Type   | Url
    |
   +--------
   +-----------------------------------------------------------------+
   | serial | ws://127.0.0.1:6083/?token=18510769-71ad-4e5a-8348-4218b5613b3d
    |
   +--------
   +-----------------------------------------------------------------+
   ```

   Alternatively, use the API directly:

   ```
   $ curl -i 'http://<controller>:8774/v2/<tenant_uuid>/servers/
   <instance_uuid>/action' \
     -X POST \
     -H "Accept: application/json" \
     -H "Content-Type: application/json" \
     -H "X-Auth-Project-Id: <project_id>" \
   ```

```
  -H "X-Auth-Token: <auth_token>" \
  -d '{"os-getSerialConsole": {"type": "serial"}}'
```

2. Use Python websocket with the URL to generate `.send`, `.recv`, and `.fileno` methods for serial console access. For example:

```
import websocket
ws = websocket.create_connection(
    'ws://127.0.0.1:6083/?token=18510769-71ad-4e5a-8348-4218b5613b3d',
    subprotocols=['binary', 'base64'])
```

   Alternatively, use a Python websocket client such as https://github.com/larsks/nova-console/.

Note

When you enable the serial console, typical instance logging using the **nova console-log** command is disabled. Kernel output and other system messages will not be visible unless you are actively viewing the serial console.

Secure with rootwrap

Rootwrap allows unprivileged users to safely run Compute actions as the root user. Compute previously used **sudo** for this purpose, but this was difficult to maintain, and did not allow advanced filters. The **rootwrap** command replaces **sudo** for Compute.

To use rootwrap, prefix the Compute command with **nova-rootwrap**. For example:

```
$ sudo nova-rootwrap /etc/nova/rootwrap.conf command
```

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. It then checks if the command requested by Compute matches one of those filters and, if so, executes the command (as root). If no filter matches, it denies the request.

Note

Be aware of issues with using NFS and root-owned files. The NFS share must be configured with the `no_root_squash` option enabled, in order for rootwrap to work correctly.

Rootwrap is fully controlled by the root user. The root user owns the sudoers entry which allows Compute to run a specific rootwrap executable as root, and only with a specific configuration file (which should also be owned by root). The **nova-rootwrap** command imports the Python modules it needs from a cleaned, system-default *PYTHONPATH*. The root-owned configuration file 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.

Rootwrap is configured using 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 in both the sudoers entry and in the `nova.conf` configuration file with the `rootwrap_config=entry` parameter.

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 rootwrap filters are stored. Directories defined on this line should all exist, and be owned and writable only by the root user.

If the root wrapper is not performing correctly, you can add a workaround option into the `nova.conf` configuration file. This workaround re-configures the root wrapper configuration to fall back to running commands as sudo, and is a Kilo release feature.

Including this workaround in your configuration file safeguards your environment from issues that can impair root wrapper performance. Tool changes that have impacted Python Build Reasonableness (PBR), for example, are a known issue that affects root wrapper performance.

To set up this workaround, configure the *`disable_rootwrap`* option in the `[workaround]` section of the `nova.conf` configuration file.

The filters definition files contain lists of filters that 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, which should be different for each filter 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 the filter class to use, followed by the 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:

- **Non-live migration** (sometimes referred to simply as 'migration'). The instance is shut down for a period of time to be moved to another hypervisor. In this case, the instance recognizes that it was rebooted.

- **Live migration** (or 'true live migration'). Almost no instance downtime. Useful when the instances must be kept running during the migration. The different 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**. Instances are backed by volumes rather than ephemeral disk, no shared storage is required, and migration is supported (currently only available for 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

Shared storage

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 “Enabling true live migration” [108].
- 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 the commands accordingly.
- You must specify `vncserver_listen=0.0.0.0` or live migration will not work correctly.
- You must specify the `instances_path` in each node that runs nova-compute. The mount point for `instances_path` must be the same value for each node, or live migration will not work correctly.

Example Compute installation environment

- Prepare at least three servers. In this example, we refer to the servers as `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 directory. HostB and HostC are NFSv4 clients that mount HostA.

Configuring 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 ensure it is readable and writable by the Compute user on HostB and HostC.

   For more information, see: SettingUpNFSHowTo or CentOS/Red Hat: 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. On both compute nodes, enable the 'execute/search' bit on your shared directory 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 on 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:

```
$ 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 on `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.

   - 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 which ports you open, and always 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 will not need to configure any options. The following chart is for advanced users only.

Table 4.6. Description of live migration configuration options

Configuration option = Default value	Description
[DEFAULT]	

Configuration option = Default value	Description
`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, VIR_MIGRATE_LIVE, VIR_MIGRATE_TUNNELLED*	(StrOpt) Migration flags to be set for live migration
`live_migration_uri` = *qemu+tcp://%s/system*	(StrOpt) Migration target URI (any included "%s" is replaced with the migration target hostname)

Enabling true live migration

Prior to the Kilo release, the Compute service did not use the libvirt live migration function by default. To enable this function, add the following line to the `[libvirt]` section of the `nova.conf` file:

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

On versions older than Kilo, the Compute service does not use libvirt's live migration by default because there is a risk that the migration process will never end. This can happen if the guest operating system uses blocks on the disk faster than they can be migrated.

Block Migration

Configuring KVM for block migration is exactly the same as the above configuration in the section called "Shared storage" [105], except that *NOVA-INST-DIR*/`instances` is local to each host rather than shared. No NFS client or server configuration is required.

Note

- To use block migration, you must use the `--block-migrate` parameter with the live migration command.
- Block migration is incompatible with read-only devices such as CD-ROMs and Configuration Drive (config_drive).
- Since the ephemeral drives are copied over the network in block migration, migrations of instances with heavy I/O loads may never complete if the drives are writing faster than the data can be copied over the network.

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 specific metadata. This metadata is used by the XAPI plug-ins to establish the pool.

Using shared storage live migration with XenServer hypervisors

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

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

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

3. Create a host aggregate. This command creates the aggregate, and then displays a table that contains the ID of the new aggregate:

   ```
   $ nova aggregate-create POOL_NAME AVAILABILITY_ZONE
   ```

   Add 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 MASTER_COMPUTE_NAME
   ```

   The host is now part of a XenServer pool.

4. Add hosts to the pool:

   ```
   $ nova aggregate-add-host AGGREGATE_ID COMPUTE_HOST_NAME
   ```

   Note

   The added compute node and the host will shut down to join the host to the XenServer pool. The operation will fail if any server other than the compute node is running or suspended on the 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 `--block-migrate` parameter with the live migration command.
- Block migration works only with EXT local storage storage repositories, and the server must not have any volumes attached.

Migrate instances

This section discusses how to migrate running instances from one OpenStack Compute server to another OpenStack Compute server.

Before starting a migration, review the Configure migrations section.

Note

Although the **nova** command is called **live-migration**, under the default Compute configuration options, the instances are suspended before migration. For more information, see Configure migrations in the *OpenStack Configuration Reference*.

Migrating instances

1. Check the ID of the instance to be migrated:

```
$ 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. Check the information associated with the instance. In this example, `vm1` is running on `HostB`:

```
$ 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
                        |
...
+-------------------------------------
+-----------------------------------------------------------+
```

3. Select the compute node the instance will be migrated to. In this example, we will migrate the instance to `HostC`, because `nova-compute` is running on it.:

```
# 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 | -               |
+------------------+------------+----------+---------+-------
+----------------------------+-----------------+
```

4. Check 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`: Number of CPUs
- `memory_mb`: Total amount of memory, in MB
- `disk_gb`: Total amount of space for NOVA-INST-DIR/instances, in GB

In this table, the first row shows the total amount of resources available on the physical server. The second line shows the currently used resources. The third line shows the maximum used resources. The fourth line and below shows the resources available for each project.

5. Migrate the instances using the **nova live-migration** command:

```
$ nova live-migration SERVER HOST_NAME
```

In this example, *SERVER* can be the ID or name of the instance. Another example:

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

Warning

When using live migration to move workloads between Icehouse and Juno compute nodes, it may cause data loss because libvirt live migration with shared block storage was buggy (potential loss of data) before version 3.32. This issue can be solved when we upgrade to RPC API version 4.0.

6. Check the instances have been migrated successfully, using **nova list**. If instances are still running on `HostB`, check the log files at src/dest for `nova-compute` and `nova-scheduler`) to determine why.

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.

Note

The web proxy console URLs do not support the websocket protocol scheme (ws://) on python versions less than 2.7.4.

The VNC console connection works as follows:

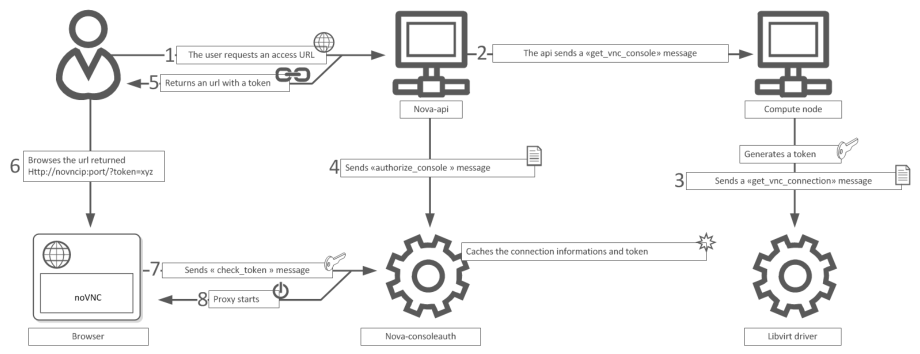
1. A user connects to the API and gets an `access_url` such as, `http://ip:port/?token=xyz`.
2. The user pastes the URL in a browser or uses it as a client parameter.
3. The browser or client connects to the proxy.
4. The proxy talks to `nova-consoleauth` to authorize the token for the user, and maps 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
5 Returns an url with a token
2 The api sends a «get_vnc_console» message
Nova-api
Compute node
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
noVNC
7 Sends « check_token » message
8 Proxy starts
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 in your `nova.conf` file:

Table 4.7. Description of VNC configuration options

Configuration option = Default value	Description
[DEFAULT]	
daemon = *False*	(BoolOpt) Become a daemon (background process)
key = *None*	(StrOpt) SSL key file (if separate from cert)
novncproxy_host = *0.0.0.0*	(StrOpt) Host on which to listen for incoming requests
novncproxy_port = *6080*	(IntOpt) Port on which to listen for incoming requests
record = *False*	(BoolOpt) Record sessions to FILE.[session_number]
source_is_ipv6 = *False*	(BoolOpt) Source is ipv6
ssl_only = *False*	(BoolOpt) Disallow non-encrypted connections
web = */usr/share/spice-html5*	(StrOpt) Run webserver on same port. Serve files from DIR.
[vmware]	
vnc_port = *5900*	(IntOpt) VNC starting port
vnc_port_total = *10000*	(IntOpt) Total number of VNC ports
[vnc]	
enabled = *True*	(BoolOpt) Enable VNC related features
keymap = *en-us*	(StrOpt) Keymap for VNC
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"
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
xvpvncproxy_base_url = *http://127.0.0.1:6081/console*	(StrOpt) Location of nova xvp VNC console proxy, in the form "http://127.0.0.1:6081/console"

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 nova-novncproxy
```

The service starts automatically on installation.

To restart the service, run:

```
# service nova-novncproxy 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 14.04, 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>
  ```

- **Q: My noVNC connections failed with ValidationError: Origin header protocol does not match. Why?**

  A: Make sure the `base_url` match your TLS setting. If you are using https console connections, make sure that the value of `novncproxy_base_url` is set explicitly where the `nova-novncproxy` service is running.

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 SPICE configuration options

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"
`html5proxy_host` = *`0.0.0.0`*	(StrOpt) Host on which to listen for incoming requests
`html5proxy_port` = *`6082`*	(IntOpt) Port on which to listen for incoming requests
`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

The Compute service must know the status of each compute node to effectively manage and use them. This can include events like a user launching a new VM, the scheduler sending a request to a live node, or a query to the ServiceGroup API to determine if a node is live.

When a compute worker running the `nova-compute` daemon starts, it calls the `join` API to join the compute group. Any service (such as 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.

Database ServiceGroup driver

By default, Compute uses the database driver to track if a node is live. 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 if a node is dead.

The driver has limitations, which can be problematic depending on your environment. If a lot of compute worker nodes need to be checked, the database can be put under heavy load, which can cause the timeout to trigger, and a live node could incorrectly be considered dead. By default, the timeout is 60 seconds. Reducing the timeout value can help in this situation, but you must also make the database update more frequently, which again increases the database workload.

The database contains data that is both transient (such as whether the node is alive) and persistent (such as 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, unlike databases, is a distributed system, with its load divided among several servers. On a compute worker node, the driver can establish a ZooKeeper session, then create 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 can be given group membership by running the **ls** command in the group directory.

The ZooKeeper driver requires the ZooKeeper servers and client libraries. Setting up ZooKeeper servers is outside the scope of this guide (for more information, see Apache Zookeeper). These client-side Python libraries must be installed on every compute node:

`python-zookeeper`	The official Zookeeper Python binding
`evzookeeper`	This library makes the binding work with the eventlet threading model.

This 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`.

These values in the `/etc/nova/nova.conf` file are required on every node for the ZooKeeper driver:

```
# Driver for the ServiceGroup service
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 these configuration option settings:

Table 4.9. Description of Zookeeper configuration options

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`, a distributed memory object caching system that is used to increase site performance. For more details, see memcached.org.

To use the `memcache` driver, you must install `memcached`. You might already have it installed, as the same driver is also used for the OpenStack Object Storage and OpenStack dashboard. If you need to install `memcached`, see the instructions in the *OpenStack Installation Guide*.

These values in the `/etc/nova/nova.conf` file are required on every node for the `memcache` driver:

```
# Driver for the ServiceGroup service
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

Administrators can designate a group of compute hosts as trusted using trusted compute pools. The trusted 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.

Trusted compute pools provide the ability for cloud subscribers to request services run only on verified compute nodes.

The remote attestation server performs node verification like this:

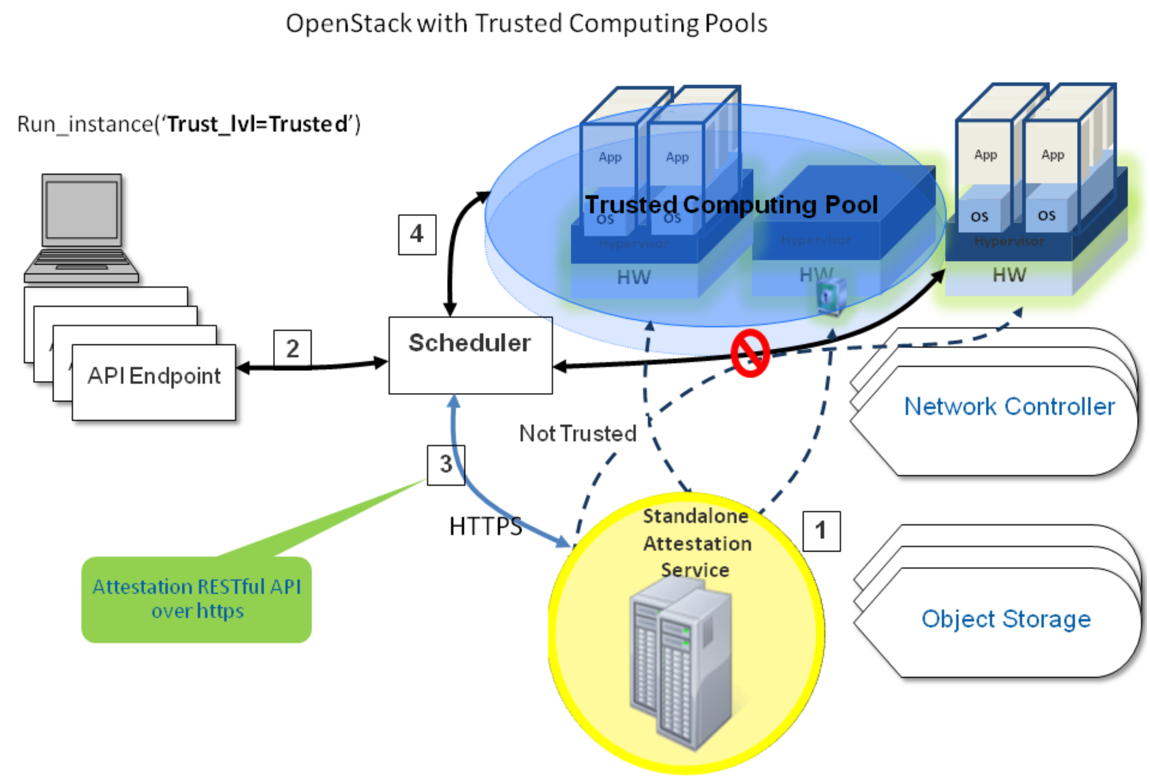
1. Compute nodes boot with Intel TXT technology enabled.

2. The compute node BIOS, hypervisor, and operating system are measured.

3. When the attestation server challenges the compute node, the measured data is sent to the attestation server.

4. The attestation server verifies the measurements against a known good 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
2
Scheduler
4
Trusted Computing Pool
App
OS
Hypervisor
HW
Not Trusted
3
HTTPS
1
Standalone Attestation Service
Network Controller
Object Storage
Attestation RESTful API over https

Configuring Compute to use trusted compute pools

1. Enable scheduling support for trusted compute pools by adding these lines to the DEFAULT section of 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 these lines to the `trusted_computing` section of the `/etc/nova/nova.conf` file:

```
[trusted_computing]
attestation_server = 10.1.71.206
attestation_port = 8443
# If using OAT v2.0 after, use this port:
# attestation_port = 8181
attestation_server_ca_file = /etc/nova/ssl.10.1.71.206.crt
# If using OAT v1.5, use this api_url:
attestation_api_url = /AttestationService/resources
# If using OAT pre-v1.5, use this api_url:
# attestation_api_url = /OpenAttestationWebServices/V1.0
attestation_auth_blob = i-am-openstack
```

In this example:

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, required by the attestation service.

3. Save the file, and restart the `nova-compute` and `nova-scheduler` services to pick up the changes.

To customize the trusted compute pools, use these configuration option settings:

Table 4.10. Description of trusted computing configuration options

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_insecure_ssl` = *`False`*	(BoolOpt) Disable SSL cert verification for Attestation service
`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

Specifying trusted flavors

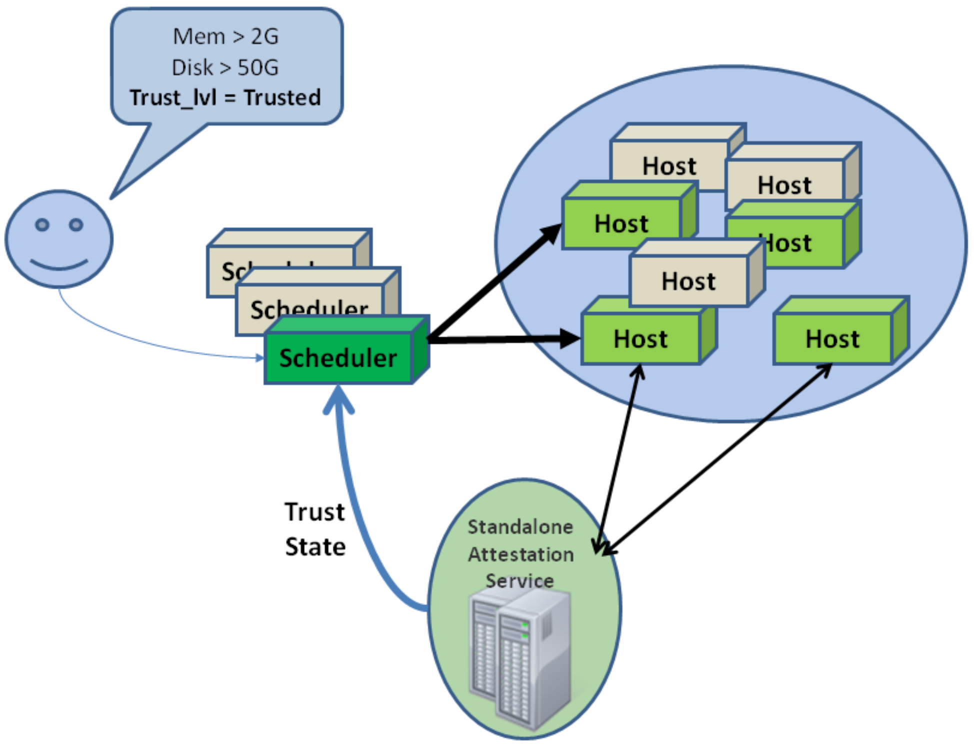
1. Flavors can be designated as trusted using the **nova flavor-key set** command. In this example, the `m1.tiny` flavor is being set as trusted:

```
$ nova flavor-key m1.tiny set trust:trusted_host=trusted
```

2. You can request that your instance is run on a trusted host by specifying a trusted flavor when booting the instance:

```
$ nova boot --flavor m1.tiny --key_name myKeypairName --image myImageID
 newInstanceName
```

Figure 4.8. Trusted compute pool



Encrypt Compute metadata traffic

OpenStack supports encrypting Compute metadata traffic with HTTPS. Enable SSL encryption in the `metadata_agent.ini` file.

Enabling SSL encryption

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 Compute is deployed with a shared file system, and a node fails, there are several methods to quickly recover from the failure. This section discusses manual recovery.

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 optionally include the target host on the **evacuate** command. If you omit the host, the scheduler determines the target host.

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 Cloud Administrator Guide*. In the following example, the password remains unchanged:

   ```
   $ nova evacuate evacuated_server_name host_b --on-shared-storage
   ```

Manual recovery

To recover a KVM or libvirt compute node, see the section called "Manual recovery" [123]. For all other hypervisors, use this procedure:

1. Identify the VMs on the affected hosts. To do this, you can use a combination of **nova list** and **nova show** or **euca-describe-instances**. For example, this command 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. Query the Compute database to check the status of the host. This example converts an EC2 API instance ID into an OpenStack ID. If you use the **nova** commands, you can substitute the ID directly (the output in this example has been truncated):

```
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
...
```

Note

The credentials for your database can be found in `/etc/nova.conf`.

3. Decide which compute host the affected VM should be moved to, and run this database command to move the VM to the new host:

```
mysql> UPDATE instances SET host = 'np-rcc46' WHERE uuid = '3f57699a-
e773-4650-a443-b4b37eed5a06';
```

4. If you are using a hypervisor that relies on libvirt (such as KVM), update the `libvirt.xml` file (found in `/var/lib/nova/instances/[instance ID]`) with these changes:

   - Change the DHCPSERVER value to the host IP address of the new compute host.
   - Update the VNC IP to `0.0.0.0`.

5. Reboot the VM:

```
$ nova reboot 3f57699a-e773-4650-a443-b4b37eed5a06
```

The database update and **nova reboot** command should be all that is required to recover a VM from a failed host. However, if you continue to have problems try 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

In some cases, files on your compute node can end up using the wrong UID or GID. This can happen when running OpenStack Compute, using a shared file system, or with an automated configuration tool. This can cause a number of problems, such as inability to perform live migrations, or start virtual machines.

This procedure runs on `nova-compute` hosts, based on the KVM hypervisor:

Recovering from a UID/GID mismatch

1. Set the nova UID in `/etc/passwd` to the same number on all hosts (for example, 112).

   Note

   Make sure you choose UIDs or GIDs that are not in use for other users or groups.

2. Set the `libvirt-qemu` UID in `/etc/passwd` to the same number on all hosts (for example, 119).

3. Set the `nova` group in `/etc/group` file to the same number on all hosts (for example, 120).

4. Set the `libvirtd` group in `/etc/group` file to the same number on all hosts (for example, 119).

5. Stop the services on the compute node.

6. Change all the files owned by user or 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 {} \;
```

7. Repeat all steps for the `libvirt-qemu` files, if required.

8. Restart the services.

9. Run the **find** command to verify that all files use the correct identifiers.

Recover cloud after disaster

This section covers procedures for managing your cloud after a disaster, and backing up persistent storage volumes. Backups are mandatory, even outside of disaster scenarios.

For a definition of a disaster recovery plan (DRP), see http://en.wikipedia.org/wiki/Disaster_Recovery_Plan.

A disaster could happen to several components of your architecture (for example, a disk crash, network loss, or a power failure). In this example, the following components are configured:

- A cloud controller (`nova-api`, `nova-objectstore`, `nova-network`)
- A compute node (`nova-compute`)
- A storage area network (SAN) used by OpenStack Block Storage (`cinder-volumes`)

The worst disaster for a cloud is power loss, which applies to all three components. Before a power loss:

- Create an active iSCSI session from the SAN to the cloud controller (used for the `cinder-volumes` LVM's VG).
- Create an active iSCSI session from the cloud controller to the compute node (managed by `cinder-volume`).
- Create an iSCSI session for every volume (so 14 EBS volumes requires 14 iSCSI sessions).
- Create iptables or ebtables rules from the cloud controller to the compute node. This allows access from the cloud controller to the running instance.
- Save the current state of the database, the current state of the running instances, and the attached volumes (mount point, volume ID, volume status, etc), at least from the cloud controller to the compute node.

After power is recovered and all hardware components have restarted:

- The iSCSI session from the SAN to the cloud no longer exists.
- The iSCSI session from the cloud controller to the compute node no longer exists.
- The iptables and ebtables from the cloud controller to the compute node are recreated. This is because `nova-network` reapplies configurations on boot.
- Instances are no longer running.

  Note that instances will not be lost, because neither **destroy** nor **terminate** was invoked. The files for the instances will remain on the compute node.
- The database has not been updated.

Begin recovery

Warning

Do not add any extra steps to this procedure, or perform the steps out of order.

1. Check the current relationship between the volume and its instance, so that you can recreate the attachment.

   This information can be found using the **nova volume-list**. Note that the **nova** client also includes the ability to get volume information from OpenStack Block Storage.

2. Update the database to clean the stalled state. Do this for every volume, using these queries:

   ```
   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;
   ```

   Use **nova volume-list** commands to list all volumes.

3. Restart the instances using the **nova reboot *INSTANCE*** command.

   Important

   Some instances will completely reboot and become reachable, while some might stop at the plymouth stage. This is expected behavior, DO NOT reboot a second time.

   Instance state at this stage 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. This step is performed in order to ask Compute to reboot every instance, so that the stored state is preserved. It does not matter if not all instances come up successfully. For more information about cloud-init, see help.ubuntu.com/community/CloudInit.

4. Reattach the volumes to their respective instances, if required, using the **nova volume-attach** command. This example 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
```

   Instances that were stopped at the plymouth stage will now automatically continue booting and start normally. Instances that previously started successfully will now be able to see the volume.

5. SSH into the instances and reboot them.

   If some services depend on the volume, or if a volume has an entry in `fstab`, you should now be able to restart the instance. Restart directly from the instance itself, not through **nova**:

```
# shutdown -r now
```

When you are planning for and performing a disaster recovery, follow these tips:

- Use the `errors=remount` parameter in the `fstab` file to prevent data corruption.

  This parameter will cause the system to disable the ability to 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), and into the instances' `fstab` files.

- 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 this command before performing the mount:

```
# iscsiadm -m discovery -t st -p $SAN_IP $ iscsiadm -m node --target-name
 $IQN -p $SAN_IP -l
```

- On 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 allows you to connect to the instance even without the volume attached, if you allow only connections through public keys.

If you want to script the disaster recovery plan (DRP), a bash script is available from https://github.com/Razique which performs the following steps:

1. An array is created for instances and their attached volumes.
2. The MySQL database is updated.
3. All instances are restarted with `euca2ools`.
4. The volumes are reattached.
5. An SSH connection is performed into every instance using Compute credentials.

The script includes a **test mode**, which allows you to perform that whole sequence for only one instance.

To reproduce the power loss, connect to the compute node which runs that instance and close the iSCSI session. Do not detach the volume using the **nova volume-detach** command, manually close the iSCSI session. This example closes an iSCSI session with the number 15:

```
# iscsiadm -m session -u -r 15
```

Do not forget the `-r` flag. Otherwise, you will 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**. Gets the `novarc` file from the project ZIP file, saves existing credentials in case of override, and manually sources 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 the following in `nova.conf`:

```
[libvirt]
inject_partition = -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.

Disable live snapshotting

If you use libvirt version `1.2.2`, you may experience problems with live snapshots creation. Occasionally, libvirt of the specified version fails to perform the live snapshotting under load that presupposes a concurrent creation of multiple snapshots.

To effectively disable the libvirt live snapshotting, until the problem is resolved, configure the `disable_libvirt_livesnapshot` option. You can turn off the live snapshotting mechanism by setting up its value to `True` in the `[workarounds]` section of the `nova.conf` configuration file:

```
[workarounds]
disable_libvirt_livesnapshot = True
```

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.

Features	Benefits
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.

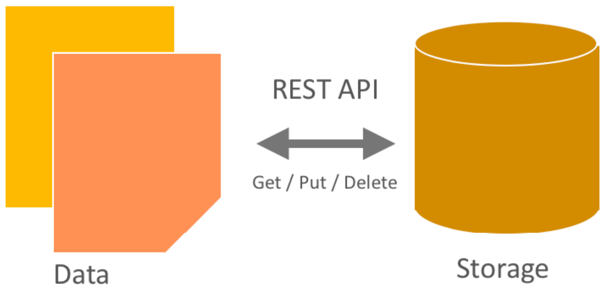
Object Storage characteristics

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)



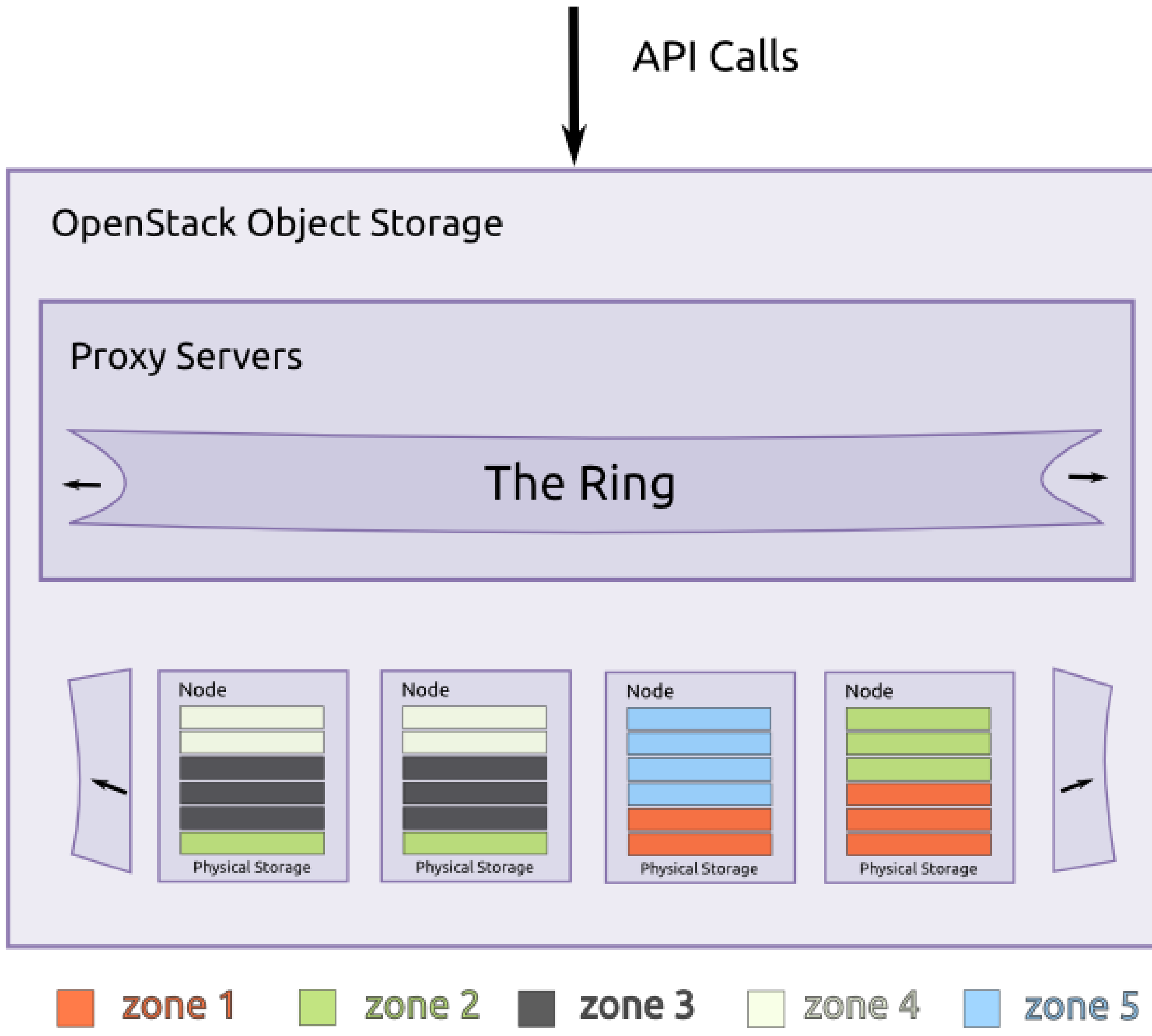
Developers can either write directly to the Swift API or use one of the many client libraries that exist for all of the popular programming languages, such as Java, Python, Ruby, and C#. Amazon S3 and RackSpace Cloud Files users should be very familiar with Object Storage. Users new to object storage 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.

For more information concerning proxy server configuration, please see the Configuration Reference.

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.

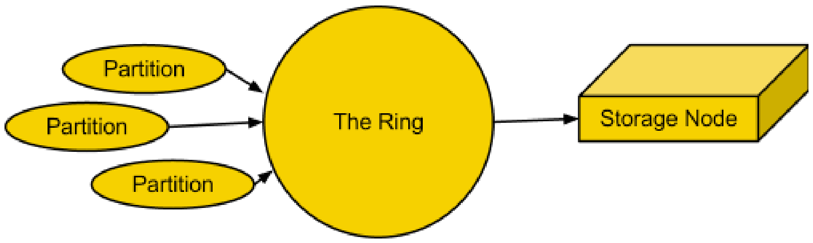
Data can be isolated into zones in the ring. Each partition replica is guaranteed to reside in a different zone. A zone could represent a drive, a server, a cabinet, a switch, or even a data center.

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.

You can use weights 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 an MD5 hash for a path 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.

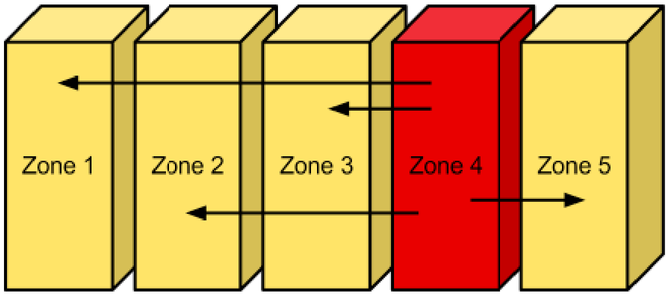
Another configurable value is the replica count, which indicates how many of the partition-device assignments make up a single ring. For a given partition number, each replica's device will not be in the same zone as any other replica's device. Zones can be used to group devices based on physical locations, 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 choosing 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

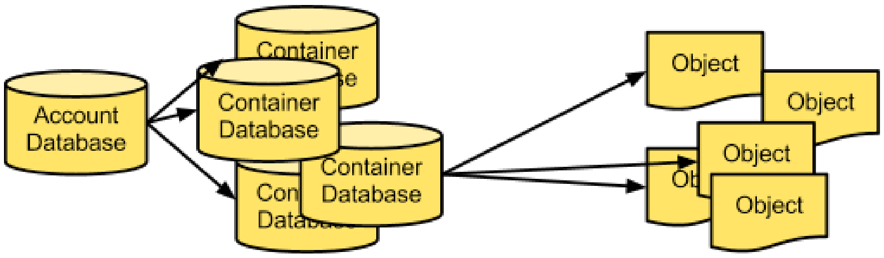


When a disk fails, replica data is automatically distributed to the other zones to ensure there are three copies of the data.

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

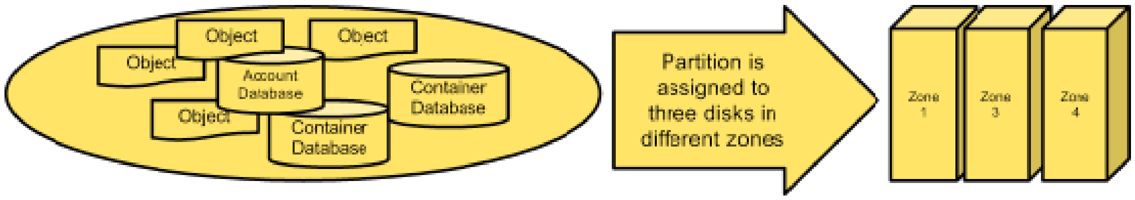
A partition is a collection of stored data, including account databases, container databases, and objects. Partitions are core to the replication system.

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

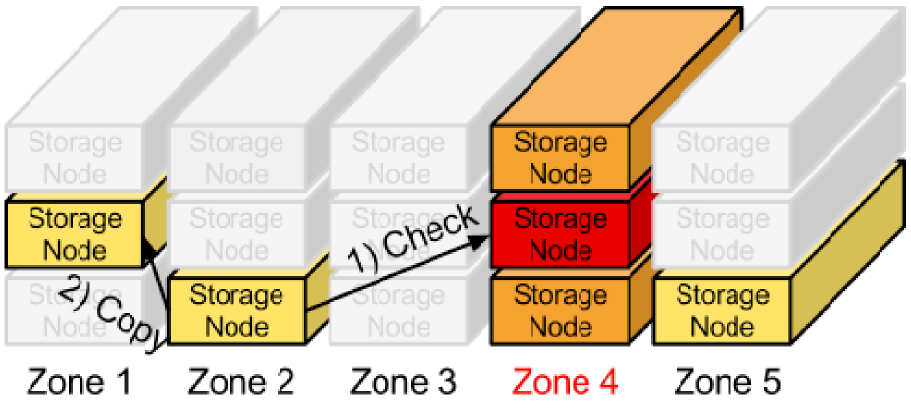
In order to ensure that there are three copies of the data everywhere, replicators continuously examine each partition. For each local partition, the replicator compares it against the replicated copies in the other zones to see if there are any differences.

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

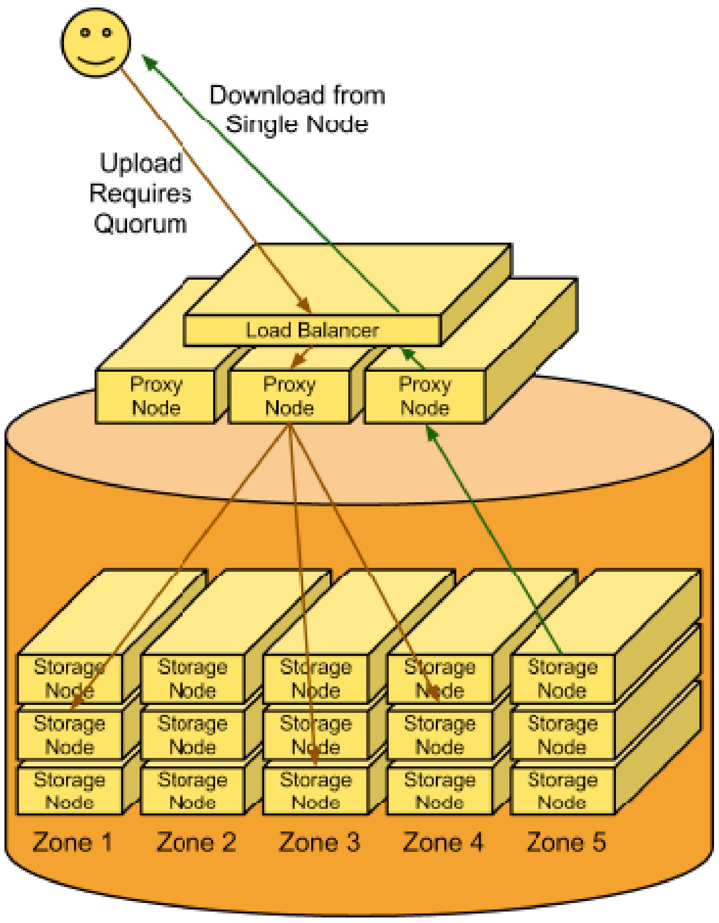
A client uses the REST API to make a HTTP request to PUT an object into an existing container. The cluster receives the request. First, the system must figure out where the data is going to go. To do this, the account name, container name, and object name are all used to determine the partition where 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 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.

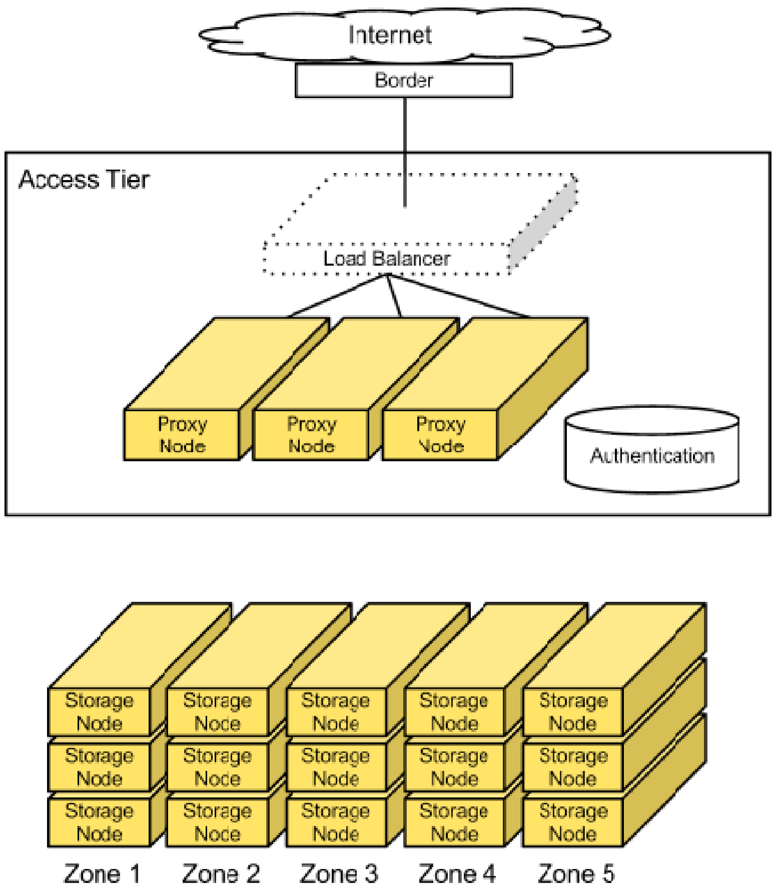
Theses steps do not always perfectly rebalance a ring due to the random nature of gathering partitions for reassignment. To help reach a more balanced ring, the rebalance process is repeated until near perfect (less than 1 percent off) or when the balance does not improve by at least 1 percent (indicating 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.

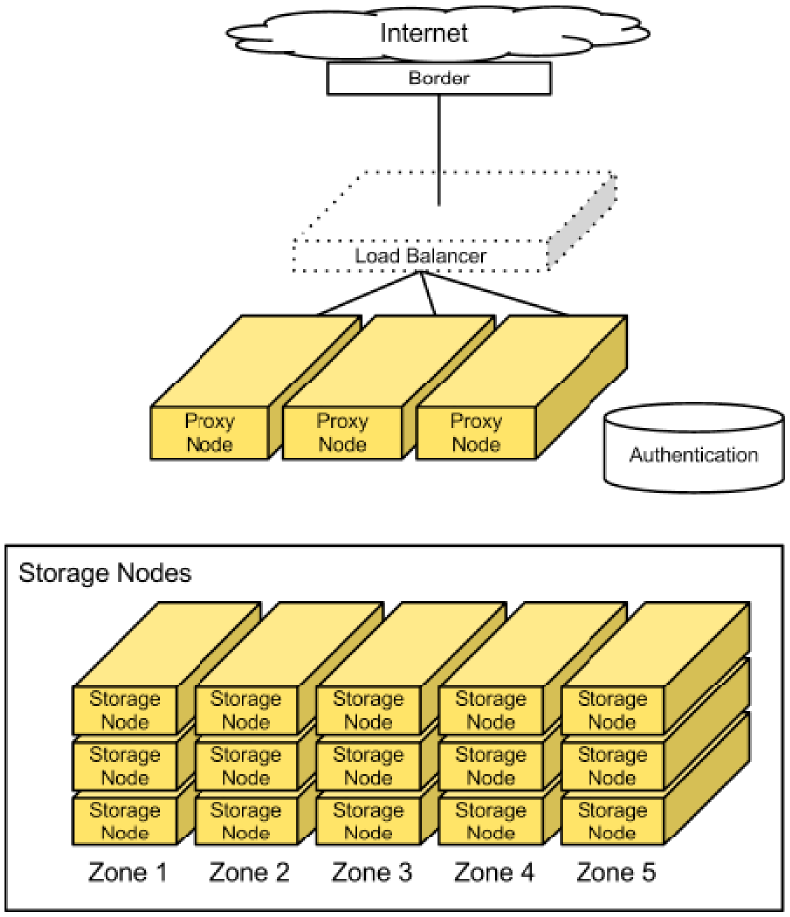
Factors to consider

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.

Account reaper

In the background, the account reaper removes data from the deleted accounts.

A reseller marks an account for deletion by issuing a `DELETE` request on the account's storage URL. This action sets the `status` column of the account_stat table in the account database and replicas to `DELETED`, marking the account's data for deletion.

Typically, a specific retention time or undelete are not provided. However, you can set a `delay_reaping` value in the `[account-reaper]` section of the account-server.conf to delay the actual deletion of data. At this time, to undelete you have to update the account database replicas directly, setting the status column to an empty string and updating the put_timestamp to be greater than the delete_timestamp.

Note

It's on the developers' to-do list to write a utility that performs this task, preferably through a REST call.

The account reaper runs on each account server and scans the server occasionally for account databases marked for deletion. It only fires up on the accounts for which the server is the primary node, so that multiple account servers aren't trying to do it simultaneously. Using multiple servers to delete one account might improve the deletion speed but requires coordination to avoid duplication. Speed really is not a big concern with data deletion, and large accounts aren't deleted often.

Deleting an account is simple. For each account container, all objects are deleted and then the container is deleted. Deletion requests that fail will not stop the overall process but will cause the overall process to fail eventually (for example, if an object delete times out, you will not be able to delete the container or the account). The account reaper keeps trying to delete an account until it is empty, at which point the database reclaim process within the db_replicator will remove the database files.

A persistent error state may prevent the deletion of an object or container. If this happens, you will see a message such as "`Account <name> has not been reaped since <date>`" in the log. You can control when this is logged with the `reap_warn_after` value in the `[account-reaper]` section of the account-server.conf file. The default value is 30 days.

Configure tenant-specific image locations with Object Storage

For some deployers, it is not ideal to store all images in one place to enable all tenants and users to access them. You can configure the Image service to store image data in tenant-specific image locations. Then, only the following tenants can use the Image service to access the created image:

- The tenant who owns the image
- Tenants that are defined in `swift_store_admin_tenants` and that have admin-level accounts

To configure tenant-specific image locations

1. Configure swift as your `default_store` in the `glance-api.conf` file.
2. Set these configuration options in the `glance-api.conf` file:
   - `swift_store_multi_tenant`. Set to `True` to enable tenant-specific storage locations. Default is `False`.
   - `swift_store_admin_tenants`. Specify a list of tenant IDs that can grant read and write access to all Object Storage containers that are created by the Image service.

With this configuration, images are stored in an Object Storage service (swift) endpoint that is pulled from the service catalog for the authenticated user.

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 meters 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 meters:

- 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 meters overlap with existing server monitoring systems. To get the Swift-specific meters into a monitoring system, they must be polled. Swift Recon essentially acts as a middleware meters collector. The process that feeds meters 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 meters directly.

Swift-Informant

Florian Hines developed the Swift-Informant middleware (see https://github.com/pandemicsyn/swift-informant) 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 meters to a StatsD server (see http://codeascraft.etsy.com/2011/02/15/measure-anything-measure-everything/):

- A counter increment for a meter like `obj.GET.200` or `cont.PUT.404`.
- Timing data for a meter like `acct.GET.200` or `obj.GET.200`. [The README says the meters 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 meters into five derivative meters with new segments appended, so it probably works as coded. The first meter 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 meter like `tfer.obj.PUT.201`.

This is good for getting a feel for the quality of service clients are experiencing with the timing meters, 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. Meter names are mapped to regex match patterns in a JSON file, allowing flexible configuration of what meters 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 meters 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 meters.

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; meters are sent in real-time by the code that just noticed or did something. The overhead of sending a meter 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 meters upstream.

To avoid the problems inherent with middleware-based monitoring and after-the-fact log processing, the sending of StatsD meters is integrated into Object Storage itself. The submitted change set (see https://review.openstack.org/#change,6058) currently reports 124 meters across 15 Object Storage daemons and the tempauth middleware. Details of the meters tracked are in the Administrator's Guide.

The sending of meters 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 meters 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 meter. The default prefix is the "name" of the logger such as "object-server", "container-auditor", and so on. 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 meter 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 meter by one.
- `decrement(self, metric, sample_rate=1)` Lowers the given counter meter by one.
- `timing(self, metric, timing_ms, sample_rate=1)` Record that the given meter took the supplied number of milliseconds.
- `timing_since(self, metric, orig_time, sample_rate=1)` Convenience method to record a timing meter 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 meter 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 meters 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.

System administration for Object Storage

By understanding Object Storage concepts, you can better monitor and administer your storage solution. The majority of the administration information is maintained in developer documentation at docs.openstack.org/developer/swift/.

See the *OpenStack Configuration Reference* for a list of configuration options for 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`

Configuration option = Default value	**Description**
`device_dir` = *`/srv/node`*	Directory devices are mounted under
`error_limit` = *`1`*	Number of errors to find before a device is unmounted
`log_address` = *`/dev/log`*	Location where syslog sends the logs to
`log_facility` = *`LOG_LOCAL0`*	Syslog log facility
`log_file_pattern` = *`/var/log/kern.*[!.][!g][!z]`*	Location of the log file with globbing pattern to check against device errors locate device blocks with errors in the log file
`log_level` = *`INFO`*	Logging level
`log_max_line_length` = *`0`*	Caps the length of log lines to the value given; no limit if set to 0, the default.
`log_to_console` = *`False`*	No help text available for this option.
`minutes` = *`60`*	Number of minutes to look back in `/var/log/kern.log`
`recon_cache_path` = *`/var/cache/swift`*	Directory where stats for a few items will be stored
`regex_pattern_1` = *`\berror\b.*\b(dm-[0-9]{1,2}\d?)\b`*	No help text available for this option.
`unmount_failed_device` = *`True`*	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
>>> from array import array
>>> builder._last_part_moves = array('B', (0 for _ in xrange(partitions)))
>>> 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
```

Next, validate the builder. If this raises an exception, check your previous code. When it validates, you're ready to save the builder and create a new account.builder.

```
>>> builder.validate()
```

Save the builder.

```

>>> import pickle
>>> pickle.dump(builder.to_dict(), open('account.builder', 'wb'), protocol=2)
>>> exit ()
```

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.

Increase Block Storage API service throughput

By default, the Block Storage API service runs in one process. This limits the number of API requests that the Block Storage service can process at any given time. In a production environment, you should increase the Block Storage API throughput by allowing the Block Storage API service to run in as many processes as the machine capacity allows.

Note

The Block Storage API service is named `openstack-cinder-api` on the following distributions: CentOS, Fedora, openSUSE, Red Hat Enterprise Linux, and SUSE Linux Enterprise. In Ubuntu and Debian distributions, the Block Storage API service is named `cinder-api`.

To do so, use the Block Storage API service option `osapi_volume_workers`. This option allows you to specify the number of API service workers (or OS processes) to launch for the Block Storage API service.

To configure this option, open the `/etc/cinder/cinder.conf` configuration file and set the `osapi_volume_workers` configuration key to the number of CPU cores/threads on a machine.

On distributions that include openstack-config, you can configure this by running the following command instead:

```
# openstack-config --set /etc/cinder/cinder.conf \
  DEFAULT osapi_volume_workers CORES
```

Replace *CORES* with the number of CPU cores/threads on a machine.

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.

To create and attach a volume to an instance

1. Configure the OpenStack Compute and the OpenStack Block Storage services through the `cinder.conf` file.

2. Use the **cinder create** command to create a volume. This command creates an LV into the volume group (VG) `cinder-volumes`.

3. Use the **nova volume-attach** command to attach the volume to an instance. 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.

The network mode does not interfere with OpenStack Block Storage operations, but you must set up networking for Block Storage to work. For details, see Chapter 7, "Networking" [203].

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, you can store and run instances 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`.

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.

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
   ```

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 Block Storage 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/cin-`

der/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, 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
   ```

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 multiple-storage back ends

When you configure multiple-storage back ends, you can create several back-end storage solutions that serve the same OpenStack Compute configuration and one `cinder-volume` is launched for each back-end storage or back-end storage pool.

In a multiple-storage 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 multiple-storage back ends

To enable a multiple-storage back ends, 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`.

Note

After setting the `enabled_backends` flag on an existing cinder service, and restarting the Block Storage services, the original `host` service is replaced with a new host service. The new service appears with a name like `host@backend`. Use:

```
$ cinder-manage volume update_host --currentname CURRENTNAME --
newname CURRENTNAME@BACKEND
```

to convert current block devices to the new hostname.

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:

```
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 `lvm-driver-3` back end.

Note

For Fiber Channel drivers that support multipath, the configuration group requires the `use_multipath_for_image_xfer=true` option. In the example below, you can see details for HP 3PAR and EMC Fiber Channel drivers.

```
[3par]
use_multipath_for_image_xfer = true
volume_driver = cinder.volume.drivers.san.hp.hp_3par_fc.
HP3PARFCDriver
volume_backend_name = 3parfc

[emc]
use_multipath_for_image_xfer = true
volume_driver = cinder.volume.drivers.emc.emc_smis_fc.
EMCSMISFCDriver
volume_backend_name = emcfc
```

Configure Block Storage scheduler multi back end

You must enable the `filter_scheduler` option to use multiple-storage back ends. The filter scheduler:

1. Filters the available back ends. By default, `AvailabilityZoneFilter`, `CapacityFilter` and `CapabilitiesFilter` are enabled.

2. Weights the previously filtered back ends. By default, the `CapacityWeigher` option is enabled. When this option is enabled, the filter scheduler assigns the highest weight to back ends with the most available capacity.

The scheduler uses filters and weights to pick the best back end to handle the request. The scheduler uses volume types to explicitly create volumes on specific back ends.

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 file systems. This usually means that those file systems 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.

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:

```
umount /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 which support its volume-type. 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 get-pools
+----------+----------------------------------------------------+
| Property |                       Value                        |
+----------+----------------------------------------------------+
|   name   |            server1@lvmstorage-1#lvmstorage-1       |
+----------+----------------------------------------------------+
+----------+----------------------------------------------------+
| Property |                      Value                         |
+----------+----------------------------------------------------+
|   name   |            server2@lvmstorage-2#lvmstorage-2       |
+----------+----------------------------------------------------+
```

Note

Only Block Storage V2 API supports **get-pools**.

You can also get available back-ends like following:

```
# cinder-manage host list
server1@lvmstorage-1    zone1
server2@lvmstorage-2    zone1
```

But it needs to add pool name in the end. For example, `server1@lvmstorage-1#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#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, Red Hat 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#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 [--incremental] VOLUME
```

Where *`VOLUME`* is the name or ID of the volume, and `incremental` is a flag that indicates whether an incremental backup should be performed.

Without the `incremental` flag, a full backup is created by default. With the `incremental` flag, an incremental backup is created.

Note

The `incremental` flag is only available for block storage API v2. You have to specify [–os-volume-api-version 2] in the **cinder** command-line interface to use this parameter.

The incremental backup is based on a parent backup which is an existing backup with the latest timestamp. The parent backup can be a full backup or an incremental backup depending on the timestamp.

Note

The first backup of a volume has to be a full backup. Attempting to do an incremental backup without any existing backups will fail.

A new configure option `backup_swift_block_size` is introduced into `cinder.conf` for the default Swift backup driver. This is the size in bytes that changes are tracked for incremental backups. The existing `backup_swift_object_size` option, the size in bytes of Swift backup objects, has to be a multiple of `backup_swift_block_size`. The default is 32768 for `backup_swift_block_size`, and the default is 52428800 for `backup_swift_object_size`.

This command also returns a backup ID. Use this backup ID when restoring the volume:

```
$ cinder backup-restore BACKUP_ID
```

When restoring from a full backup, it is a full restore.

When restoring from an incremental backup, a list of backups is built based on the IDs of the parent backups. A full restore is performed based on the full backup first, then restore is done based on the incremental backup, laying on top of it in order.

Because volume backups are dependent on the Block Storage database, you must also back up your Block Storage database regularly to ensure data recovery.

Note

Alternatively, you can export and save the metadata of selected volume backups. Doing so precludes the need to back up the entire Block Storage database. This is useful if you need only a small subset of volumes to survive a catastrophic database failure.

If you specify a UUID encryption key when setting up the volume specifications, the backup metadata ensures that the key will remain valid when you back up and restore the volume.

For more information about how to export and import volume backup metadata, see the section called "Export and import backup metadata" [177].

By default, the swift object store is used for the backup repository.

If instead you want to use an NFS export as the backup repository, add the following configuration options to the `[DEFAULT]` section of the `cinder.conf` file and restart the Block Storage services:

```
backup_driver = cinder.backup.drivers.nfs
backup_share = HOST:EXPORT_PATH
```

For the `backup_share` option, replace *`HOST`* with the DNS resolvable host name or the IP address of the storage server for the NFS share, and *`EXPORT_PATH`* with the path to that share. If your environment requires that non-default mount options be specified for the share, set these as follows:

```
backup_mount_options = MOUNT_OPTIONS
```

`MOUNT_OPTIONS` is a comma-separated string of NFS mount options as detailed in the NFS man page.

There are several other options whose default values may be overriden as appropriate for your environment:

```
backup_compression_algorithm = zlib
backup_sha_block_size_bytes = 32768
backup_file_size = 1999994880
```

The option `backup_compression_algorithm` can be set to `bz2` or `None`. The latter can be a useful setting when the server providing the share for the backup repository itself performs deduplication or compression on the backup data.

The option `backup_file_size` must be a multiple of `backup_sha_block_size_bytes`. It is effectively the maximum file size to be used, given your environment, to hold backup data. Volumes larger than this will be stored in multiple files in the backup repository. The `backup_sha_block_size_bytes` option determines the size of blocks from the cinder volume being backed up on which digital signatures are calculated in order to enable incremental backup capability.

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 about how to back up and restore a volume, see the section called "Back up and restore volumes" [175].

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.

If you have placed encryption on your volumes, the encryption will still be in place when you restore the volume if a UUID encryption key is specified when creating volumes. Using backup metadata support, UUID keys set up for a volume (or volumes) will remain valid when you restore a backed-up volume. The restored volume will remain encrypted, and will be accessible with your credentials.

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" [175]).

Use LIO iSCSI support

The default `iscsi_helper` tool is `tgtadm`. To use LIO iSCSI, install the `python-rtslib` package, and set `iscsi_helper=lioadm` in the `cinder.conf` file.

Once configured, you can use the **cinder-rtstool** command to manage the volumes. This command enables you to create, delete, and verify volumes and determine targets and add iSCSI initiators to the system.

Configure and use volume number weigher

OpenStack Block Storage enables you to choose a volume back end according to `free_capacity` and `allocated_capacity`. The volume number weigher feature lets the scheduler choose a volume back end based on its volume number in the volume back end. This can provide another means to improve the volume back ends' I/O balance and the volumes' I/O performance.

Enable volume number weigher

To enable a volume number weigher, set the `scheduler_default_weighers` to `VolumeNumberWeigher` flag in the `cinder.conf` file to define `VolumeNumberWeigher` as the selected weigher.

Configure multiple-storage back ends

To configure `VolumeNumberWeigher`, use `LVMISCSIDriver` as the volume driver.

This configuration defines two LVM volume groups: `stack-volumes` with 10 GB capacity and `stack-volumes-1` with 60 GB capacity. This example configuration defines two back ends:

```
scheduler_default_weighers=VolumeNumberWeigher
enabled_backends=lvmdriver-1,lvmdriver-2
[lvmdriver-1]
volume_group=stack-volumes
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI

[lvmdriver-2]
volume_group=stack-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
```

Volume type

Define a volume type in Block Storage:

```
$ cinder type-create lvm
```

Create an extra specification that links the volume type to a back-end name:

```
$ cinder 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.

Usage

To create six 1-GB volumes, run the **cinder create –volume-type lvm 1** command six times:

```
$ cinder create --volume-type lvm 1
```

This command creates three volumes in `stack-volumes` and three volumes in `stack-volumes-1`.

List the available volumes:

```
# lvs
  LV                                          VG              Attr      LSize
 Pool Origin Data%  Move Log Copy%  Convert
  volume-3814f055-5294-4796-b5e6-1b7816806e5d stack-volumes   -wi-a----  1.00g
  volume-72cf5e79-99d2-4d23-b84e-1c35d3a293be stack-volumes   -wi-a----  1.00g
  volume-96832554-0273-4e9d-902b-ad421dfb39d1 stack-volumes   -wi-a----  1.00g
  volume-169386ef-3d3e-4a90-8439-58ceb46889d9 stack-volumes-1 -wi-a----  1.00g
  volume-460b0bbb-d8a0-4bc3-9882-a129a5fe8652 stack-volumes-1 -wi-a----  1.00g
  volume-9a08413b-0dbc-47c9-afb8-41032ab05a41 stack-volumes-1 -wi-a----  1.00g
```

Consistency Groups

Consistency group support is available in OpenStack Block Storage. The support is added for creating snapshots of consistency groups. This feature leverages the storage level consistency technology. It allows snapshots of multiple volumes in the same consistency group to be taken at the same point-in-time to ensure data consistency. The consistency group operations can be performed using the Block Storage command line.

Note

Only Block Storage V2 API supports consistency groups. You can specify `--os-volume-api-version 2` when using Block Storage command line for consistency group operations.

Before using consistency groups, make sure the Block Storage driver that you are running has consistency group support by reading the Block Storage manual or consulting the driver maintainer. There are a small number of drivers that have implemented this feature. The default LVM driver does not support consistency groups yet because the consistency technology is not available at the storage level.

Before using consistency groups, you must change policies for the consistency group APIs in the `/etc/cinder/policy.json` file. By default, the consistency group APIs are disabled. Enable them before running consistency group operations.

Here are existing policy entries for consistency groups:

```
"consistencygroup:create": "group:nobody",
"consistencygroup:delete": "group:nobody",
"consistencygroup:get": "group:nobody",
"consistencygroup:get_all": "group:nobody",
"consistencygroup:create_cgsnapshot" : "group:nobody",
"consistencygroup:delete_cgsnapshot": "group:nobody",
"consistencygroup:get_cgsnapshot": "group:nobody",
"consistencygroup:get_all_cgsnapshots": "group:nobody",
```

Change them to the following by removing `group:nobody` to enable these APIs:

```
"consistencygroup:create": "",
"consistencygroup:delete": "",
"consistencygroup:update": "",
"consistencygroup:get": "",
"consistencygroup:get_all": "",
"consistencygroup:create_cgsnapshot" : "",
"consistencygroup:delete_cgsnapshot": "",
"consistencygroup:get_cgsnapshot": "",
"consistencygroup:get_all_cgsnapshots": "",
```

Restart Block Storage API service after changing policies.

The following consistency group operations are supported:

- Create a consistency group, given volume types.

  Note

  A consistency group can support more than one volume type. The scheduler is responsible for finding a back end that can support all given volume types.

  Note

  A consistency group can only contain volumes hosted by the same back end.

  Note

  A consistency group is empty upon its creation. Volumes need to be created and added to it later.

- Show a consistency group.
- List consistency groups.
- Create a volume and add it to a consistency group, given volume type and consistency group id.
- Create a snapshot for a consistency group.
- Show a snapshot of a consistency group.
- List consistency group snapshots.
- Delete a snapshot of a consistency group.
- Delete a consistency group.
- Modify a consistency group.
- Create a consistency group from the snapshot of another consistency group.

The following operations are not allowed if a volume is in a consistency group:

- Volume migration.
- Volume retype.

- Volume deletion.

Note

A consistency group has to be deleted as a whole with all the volumes.

The following operations are not allowed if a volume snapshot is in a consistency group snapshot:

- Volume snapshot deletion.

Note

A consistency group snapshot has to be deleted as a whole with all the volume snapshots.

The details of consistency group operations are shown in the following.

Create a consistency group:

```
cinder consisgroup-create
[--name name]
[--description description]
[--availability-zone availability-zone]
volume-types
```

Note

The parameter `volume-types` is required. It can be a list of names or UUIDs of volume types separated by commas without spaces in between. For example, `volumetype1,volumetype2,volumetype3.`.

```
$ cinder consisgroup-create --name bronzeCG2 volume_type_1
```

```
+-------------------+--------------------------------------+
|      Property     |                Value                 |
+-------------------+--------------------------------------+
| availability_zone |                 nova                 |
|     created_at    |      2014-12-29T12:59:08.000000      |
|    description    |                 None                 |
|         id        | 1de80c27-3b2f-47a6-91a7-e867cbe36462 |
|        name       |              bronzeCG2               |
|       status      |               creating               |
+-------------------+--------------------------------------+
```

Show a consistency group:

```
$ cinder consisgroup-show 1de80c27-3b2f-47a6-91a7-e867cbe36462
+-------------------+--------------------------------------+
|      Property     |                Value                 |
+-------------------+--------------------------------------+
| availability_zone |                 nova                 |
|     created_at    |      2014-12-29T12:59:08.000000      |
|    description    |                 None                 |
|         id        | 2a6b2bda-1f43-42ce-9de8-249fa5cbae9a |
|        name       |              bronzeCG2               |
|       status      |              available               |
+-------------------+--------------------------------------+
```

List consistency groups:

```
$ cinder consisgroup-list
+--------------------------------------+-----------+-----------+
|                  ID                  |   Status  |    Name   |
+--------------------------------------+-----------+-----------+
| 1de80c27-3b2f-47a6-91a7-e867cbe36462 | available | bronzeCG2 |
| 3a2b3c42-b612-479a-91eb-1ed45b7f2ad5 |   error   |  bronzeCG |
+--------------------------------------+-----------+-----------+
```

Create a volume and add it to a consistency group:

Note

When creating a volume and adding it to a consistency group, a volume type and a consistency group id must be provided. This is because a consistency group can support more than one volume type.

```
$ cinder create --volume-type volume_type_1 --name cgBronzeVol --consisgroup-
id 1de80c27-3b2f-47a6-91a7-e867cbe36462 1
+---------------------------------------
+---------------------------------------+
|                Property                |                 Value
   |
+---------------------------------------
+---------------------------------------+
|              attachments               |                   []
   |
|           availability_zone            |                  nova
   |
|                bootable                |                 false
   |
|          consistencygroup_id           |  1de80c27-3b2f-47a6-91a7-
e867cbe36462  |
|               created_at               |      2014-12-29T13:16:47.000000
   |
|              description               |                  None
   |
|               encrypted                |                 False
   |
|                   id                   |  5e6d1386-4592-489f-
a56b-9394a81145fe  |
|                metadata                |                   {}
   |
|                  name                  |              cgBronzeVol
   |
|         os-vol-host-attr:host          |      server-1@backend-1#pool-1
   |
|     os-vol-mig-status-attr:migstat     |                  None
   |
|     os-vol-mig-status-attr:name_id     |                  None
   |
|      os-vol-tenant-attr:tenant_id      |    1349b21da2a046d8aa5379f0ed447bed
   |
|   os-volume-replication:driver_data    |                  None
   |
| os-volume-replication:extended_status |                  None
   |
|           replication_status           |                 disabled
   |
|                  size                  |                    1
   |
|               snapshot_id              |                   None
   |
|               source_volid             |                   None
   |
|                 status                 |                 creating
   |
|                user_id                 |     93bdea12d3e04c4b86f9a9f172359859
   |
|               volume_type              |               volume_type_1
   |
+---------------------------------------
+----------------------------------------+
```

Create a snapshot for a consistency group:

```
$ cinder cgsnapshot-create 1de80c27-3b2f-47a6-91a7-e867cbe36462
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
| consistencygroup_id | 1de80c27-3b2f-47a6-91a7-e867cbe36462 |
|      created_at     |      2014-12-29T13:19:44.000000      |
|     description     |                 None                 |
|          id         | d4aff465-f50c-40b3-b088-83feb9b349e9 |
|         name        |                 None                 |
|        status       |               creating               |
+---------------------+--------------------------------------+
```

Show a snapshot of a consistency group:

```
$ cinder cgsnapshot-show d4aff465-f50c-40b3-b088-83feb9b349e9
```

List consistency group snapshots:

```
$ cinder cgsnapshot-list
+--------------------------------------+--------+----------+
|                  ID                  | Status | Name     |
+--------------------------------------+--------+----------+
| 6d9dfb7d-079a-471e-b75a-6e9185ba0c38 | available  | None |
| aa129f4d-d37c-4b97-9e2d-7efffda29de0 | available  | None |
| bb5b5d82-f380-4a32-b469-3ba2e299712c | available  | None |
| d4aff465-f50c-40b3-b088-83feb9b349e9 | available  | None |
+--------------------------------------+--------+----------+
```

Delete a snapshot of a consistency group:

```
$ cinder cgsnapshot-delete d4aff465-f50c-40b3-b088-83feb9b349e9
```

Delete a consistency group:

Note

The force flag is needed when there are volumes in the consistency group.

```
$ cinder consisgroup-delete --force 1de80c27-3b2f-47a6-91a7-e867cbe36462
```

Modify a consistency group:

```
cinder consisgroup-update
[--name NAME]
[--description DESCRIPTION]
[--add-volumes UUID1,UUID2,......]
[--remove-volumes UUID3,UUID4,......]
CG
```

The parameter *CG* is required. It can be a name or UUID of a consistency group. *UUID1,UUID2,......* are UUIDs of one or more volumes to be added to the consistency group, separated by commas. Default is None. *UUID3,UUId4,......* are UUIDs of one or more volumes to be removed from the consistency group, separated by commas. Default is None.

```
$ cinder consisgroup-update --name 'new name' --description 'new
 description' --add-volumes 0b3923f5-95a4-4596-a536-914c2c84e2db,
1c02528b-3781-4e32-929c-618d81f52cf3 --remove-volumes 8c0f6ae4-
efb1-458f-a8fc-9da2afcc5fb1,a245423f-bb99-4f94-8c8c-02806f9246d8
 1de80c27-3b2f-47a6-91a7-e867cbe36462
```

Create a consistency group from the snapshot of another consistency group:

```
cinder consisgroup-create-from-src
[--cgsnapshot CGSNAPSHOT]
[--name NAME]
[--description DESCRIPTION]
```

The parameter *CGSNAPSHOT* is a name or UUID of a snapshot of a consistency group.

```
$ cinder consisgroup-create-from-src --cgsnapshot 6d9dfb7d-079a-471e-
b75a-6e9185ba0c38 --name 'new cg' --description 'new cg from cgsnapshot'
```

Configure and use driver filter and weighing for scheduler

OpenStack Block Storage enables you to choose a volume back end based on back-end specific properties by using the DriverFilter and GoodnessWeigher for the scheduler. The driver filter and weigher scheduling can help ensure that the scheduler chooses the best back end based on requested volume properties as well as various back-end specific properties.

What is driver filter and weigher and when to use it

The driver filter and weigher gives you the ability to more finely control how the OpenStack Block Storage scheduler chooses the best back end to use when handling a volume request. One example scenario where using the driver filter and weigher can be if a back end that utilizes thin-provisioning is used. The default filters use the "free capacity" property to determine the best back end, but that is not always perfect. If a back end has the ability to provide a more accurate back-end specific value you can use that as part of the weighing. Another example of when the driver filter and weigher can prove useful is if a back end exists where there is a hard limit of 1000 volumes. The maximum volume size is 500 GB. Once 75% of the total space is occupied the performance of the back end degrades. The driver filter and weigher can provide a way for these limits to be checked for.

Enable driver filter and weighing

To enable the driver filter, set the `scheduler_default_filters` option in the `cinder.conf` file to `DriverFilter` or add it to the list if other filters are already present.

To enable the goodness filter as a weigher, set the `scheduler_default_weighers` option in the `cinder.conf` file to `GoodnessWeigher` or add it to the list if other weighers are already present.

You can choose to use the `DriverFilter` without the `GoodnessWeigher` or vice-versa. The filter and weigher working together, however, create the most benefits when helping the scheduler choose an ideal back end.

Important

The support for the `DriverFilter` and `GoodnessWeigher` is optional for back ends. If you are using a back end that does not support the filter and weigher functionality you may not get the full benefit.

Example `cinder.conf` configuration file:

```
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
```

Note

It is useful to use the other filters and weighers available in OpenStack in combination with these custom ones. For example, the `CapacityFilter` and `CapacityWeigher` can be combined with these.

Defining your own filter and goodness functions

You can define your own filter and goodness functions through the use of various properties that OpenStack Block Storage has exposed. Properties exposed include information about the volume request being made, volume_type settings, and back-end specific information about drivers. All of these allow for a lot of control over how the ideal back end for a volume request will be decided.

The `filter_function` option is a string defining an equation that will determine whether a back end should be considered as a potential candidate in the scheduler.

The `goodness_function` option is a string defining an equation that will rate the quality of the potential host (0 to 100, 0 lowest, 100 highest).

Important

Default values for the filter and goodness functions will be used for each back end if you do not define them yourself. If complete control is desired then a filter and goodness function should be defined for each of the back ends in the `cinder.conf` file.

Supported operations in filter and goodness functions

Below is a table of all the operations currently usable in custom filter and goodness functions created by you:

Operations	Type
+, -, *, /, ^	standard math

Operations	Type
not, and, or, &, |, !	logic
>, >=, <, <=, ==, <>, !=	equality
+, -	sign
x ? a : b	ternary
abs(x), max(x, y), min(x, y)	math helper functions

Caution

Syntax errors in filter or goodness strings defined by you will cause errors to be thrown at volume request time.

Available properties when creating custom functions

There are various properties that can be used in either the `filter_function` or the `goodness_function` strings. The properties allow access to volume info, qos settings, extra specs, and so on.

Here is a list of the properties and their sub-properties currently available for use:

- `stats`—These are the host stats for a back end.

host	The host's name.
volume_backend_name	The volume back end name.
vendor_name	The vendor name.
driver_version	The driver version.
storage_protocol	The storage protocol.
QoS_support	Boolean signifying whether QoS is supported.
total_capacity_gb	The total capacity in GB.
allocated_capacity_gb	The allocated capacity in GB.
reserved_percentage	The reserved storage percentage.

- `capabilities`—These are the capabilities specific to a back end.

  The properties available here are determined by the specific back end you are creating filter and goodness functions for. Some back ends may not have any properties available here.

- `volume`—The requested volume properties.

status	Status for the requested volume.
volume_type_id	The volume type ID.
display_name	The display name of the volume.

volume_metadata	Any metadata the volume has.
reservations	Any reservations the volume has.
user_id	The volume's user ID.
attach_status	The attach status for the volume.
display_description	The volume's display description.
id	The volume's ID.
replication_status	The volume's replication status.
snapshot_id	The volume's snapshot ID.
encryption_key_id	The volume's encryption key ID.
source_volid	The source volume ID.
volume_admin_metadata	Any admin metadata for this volume.
source_replicaid	The source replication ID.
consistencygroup_id	The consistency group ID.
size	The size of the volume in GB.
metadata	General metadata.

The property most used from here will most likely be the `size` sub-property.

- `extra`—The extra specs for the requested volume type.

  View the available properties for volume types by running:

  ```
  $ cinder extra-specs-list
  ```

- `qos`—The current QoS specs for the requested volume type.

  View the available properties for volume types by running:

  ```
  $ cinder qos-list
  ```

In order to access these properties in a custom string use the following format:

<property>.<sub_property>

Driver filter and weigher usage examples

Below are examples for using the filter and weigher separately, together, and using driver-specific properties.

Example `cinder.conf` file configuration for customizing the filter function:

```
[default]
scheduler_default_filters = DriverFilter
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "volume.size < 10"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "volume.size >= 10"
```

The above example will filter volumes to different back ends depending on the size of the requested volume. Default OpenStack Block Storage scheduler weighing is done. Volumes with a size less than 10 GB are sent to lvm-1 and volumes with a size greater than or equal to 10 GB are sent to lvm-2.

Example `cinder.conf` file configuration for customizing the goodness function:

```
[default]
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
goodness_function = "(volume.size < 5) ? 100 : 50"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
goodness_function = "(volume.size >= 5) ? 100 : 25"
```

The above example will determine the goodness rating of a back end based off of the requested volume's size. Default OpenStack Block Storage scheduler filtering is done. The example shows how the ternary if statement can be used in a filter or goodness function. If a requested volume is of size 10 GB then lvm-1 is rated as 50 and lvm-2 is rated as 100. In this case lvm-2 wins. If a requested volume is of size 3 GB then lvm-1 is rated 100 and lvm-2 is rated 25. In this case lvm-1 would win.

Example `cinder.conf` file configuration for customizing both the filter and goodness functions:

```
[default]
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1, lvm-2

[lvm-1]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_backend_name = sample_LVM
filter_function = "stats.total_capacity_gb < 500"
goodness_function = "(volume.size < 25) ? 100 : 50"

[lvm-2]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
```

```
volume_backend_name = sample_LVM
filter_function = "stats.total_capacity_gb >= 500"
goodness_function = "(volume.size >= 25) ? 100 : 75"
```

The above example combines the techniques from the first two examples. The best back end is now decided based off of the total capacity of the back end and the requested volume's size.

Example `cinder.conf` file configuration for accessing driver specific properties:

```
[default]
scheduler_default_filters = DriverFilter
scheduler_default_weighers = GoodnessWeigher
enabled_backends = lvm-1,lvm-2,lvm-3

[lvm-1]
volume_group = stack-volumes-lvmdriver-1
volume_driver = cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name = lvmdriver-1
filter_function = "volume.size < 5"
goodness_function = "(capabilities.total_volumes < 3) ? 100 : 50"

[lvm-2]
volume_group = stack-volumes-lvmdriver-2
volume_driver = cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name = lvmdriver-2
filter_function = "volumes.size < 5"
goodness_function = "(capabilities.total_volumes < 8) ? 100 : 50"

[lvm-3]
volume_group = stack-volumes-lvmdriver-3
volume_driver = cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name = lvmdriver-3
goodness_function = "55"
```

The above is an example of how back-end specific properties can be used in the filter and goodness functions. In this example the LVM driver's 'total_volumes' capability is being used to determine which host gets used during a volume request. In the above example, lvm-1 and lvm-2 will handle volume requests for all volumes with a size less than 5 GB. The lvm-1 host will have priority until it contains three or more volumes. After than lvm-2 will have priority until it contains eight or more volumes. The lvm-3 will collect all volumes greater or equal to 5 GB as well as all volumes once lvm-1 and lvm-2 lose priority.

Rate-limit volume copy bandwidth

When you create a new volume from an image or an existing volume, or when you upload a volume image to the Image Service, large data copy may stress disk and network bandwidth. To mitigate slow down of data access from the instances, OpenStack Block Storage supports rate-limiting of volume data copy bandwidth.

Configure volume copy bandwidth limit

To configure the volume copy bandwidth limit, set the `volume_copy_bps_limit` option in the configuration groups for each back end in the `cinder.conf` file. This option takes the integer of maximum bandwidth allowed for volume data copy in byte per second. If this option is set to `0`, the rate-limit is disabled.

While multiple volume data copy operations are running in the same back end, the specified bandwidth is divided to each copy.

Example `cinder.conf` configuration file to limit volume copy bandwidth of `lvmdriver-1` up to 100 MiB/s:

```
[lvmdriver-1]
volume_group=cinder-volumes-1
volume_driver=cinder.volume.drivers.lvm.LVMISCSIDriver
volume_backend_name=LVM_iSCSI
volume_copy_bps_limit=104857600
```

Note

This feature requires libcgroup to set up blkio cgroup for disk I/O bandwidth limit. The libcgroup is provided by the cgroup-bin package in Debian and Ubuntu, or by the libcgroup-tools package in Fedora, Red Hat Enterprise Linux, CentOS, openSUSE, and SUSE Linux Enterprise.

Note

Some back ends which use remote file systems such as NFS are not supported by this feature.

Oversubscription in thin provisioning

OpenStack Block Storage enables you to choose a volume back end based on virtual capacities for thin provisioning using the oversubscription ratio.

A reference implementation is provided for the default LVM driver. The illustration below uses the LVM driver as an example.

Configure oversubscription settings

To support oversubscription in thin provisioning, a flag `max_over_subscription_ratio` is introduced into `cinder.conf`. This is a float representation of the oversubscription ratio when thin provisioning is involved. Default ratio is 20.0, meaning provisioned capacity can be 20 times of the total physical capacity. A ratio of 10.5 means provisioned capacity can be 10.5 times of the total physical capacity. A ratio of 1.0 means provisioned capacity cannot exceed the total physical capacity. A ratio lower than 1.0 is ignored and the default value is used instead.

Note

`max_over_subscription_ratio` can be configured for each back end when multiple-storage back ends are enabled. It is provided as a reference implementation and is used by the LVM driver. However, it is not a requirement for a driver to use this option from `cinder.conf`. `max_over_subscription_ratio` is for configuring a back end. For a driver that supports multiple pools per back end, it can report this ratio for each pool. The LVM driver does not support multiple pools.

The existing `reserved_percentage` flag is used to prevent over provisioning. This flag represents the percentage of the back-end capacity that is reserved.

Note

There is a change on how `reserved_percentage` is used. It was measured against the free capacity in the past. Now it is measured against the total capacity.

Capabilities

Drivers can report the following capabilities for a back end or a pool:

```
thin_provisioning_support=True(or False)
thick_provisioning_support=True(or False)
provisioned_capacity_gb=PROVISIONED_CAPACITY
max_over_subscription_ratio=MAX_RATIO
```

Where *`PROVISIONED_CAPACITY`* is the apparent allocated space indicating how much capacity has been provisioned and *`MAX_RATIO`* is the maximum oversubscription ratio. For the LVM driver, it is `max_over_subscription_ratio` in `cinder.conf`.

Two capabilities are added here to allow a back end or pool to claim support for thin provisioning, or thick provisioning, or both.

The LVM driver reports `thin_provisioning_support=True` and `thick_provisioning_support=False` if the `lvm_type` flag in `cinder.conf` is `thin`. Otherwise it reports `thin_provisioning_support=False` and `thick_provisioning_support=True`.

Volume type extra specs

If volume type is provided as part of the volume creation request, it can have the following extra specs defined:

```
'capabilities:thin_provisioning_support': '<is> True' or '<is> False'
'capabilities:thick_provisioning_support': '<is> True' or '<is> False'
```

Note

`capabilities` scope key before `thin_provisioning_support` and `thick_provisioning_support` is not required. So the following works too:

```
'thin_provisioning_support': '<is> True' or '<is> False'
'thick_provisioning_support': '<is> True' or '<is> False'
```

The above extra specs are used by the scheduler to find a back end that supports thin provisioning, thick provisioning, or both to match the needs of a specific volume type.

Capacity filter

In the capacity filter, `max_over_subscription_ratio` is used when choosing a back end if `thin_provisioning_support` is True and `max_over_subscription_ratio` is greater than 1.0.

Capacity weigher

In the capacity weigher, virtual free capacity is used for ranking if `thin_provisioning_support` is True. Otherwise, real free capacity will be used as before.

Troubleshoot your installation

This section provides useful tips to help you 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 failures. 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 the installer should handle all this, 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
```

Addressing discrepancies in reported volume sizes for EqualLogic storage

Problem

There is a discrepancy between both the actual volume size in EqualLogic (EQL) storage and the image size in the Image service, with what is reported OpenStack database. This could lead to confusion if a user is creating volumes from an image that was uploaded from an EQL volume (through the Image service). The image size is slightly larger than the target volume size; this is because EQL size reporting accounts for additional storage used by EQL for internal volume metadata.

To reproduce the issue follow the steps in the following procedure.

This procedure assumes that the EQL array is provisioned, and that appropriate configuration settings have been included in `/etc/cinder/cinder.conf` to connect to the EQL array.

1. Create a new volume. Note the ID and size of the volume. In the following example, the ID and size are `74cf9c04-4543-47ae-a937-a9b7c6c921e7` and `1`, respectively:

   ```
   $ cinder create --display-name volume1 1
   +-----------------------+-------------------------------------------+
   |   Property            |   Value                                   |
   +-----------------------+-------------------------------------------+
   |   attachments         |   []                                      |
   |   availability zone   |   nova                                    |
   |   bootable            |   false                                   |
   |   created_at          |   2014-03-21T18:31:54.248775              |
   |   display_description |   None                                    |
   |   display_name        |   volume1                                 |
   |   id                  |   74cf9c04-4543-47ae-a937-a9b7c6c921e7    |
   |   metadata            |   {}                                      |
   |   size                |   1                                       |
   |   snapshot_id         |   None                                    |
   |   source volid        |   None                                    |
   |   status              |   creating                                |
   |   volume type         |   None                                    |
   +-------------------------------+-----------------------------------+
   ```

2. Verify the volume size on the EQL array by using its command-line interface.

The actual size (`VolReserve`) is 1.01 GB. The EQL Group Manager should also report a volume size of 1.01 GB.

```
eql> volume select volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
eql (volume_volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7)> show
_______________________________ Volume Information
 ________________________________
Name: volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
Size: 1GB
VolReserve: 1.01GB
VolReservelnUse: 0MB
ReplReservelnUse: 0MB
iSCSI Alias: volume-74cf9c04-4543-47ae-a937-a9b7c6c921e7
iSCSI Name: iqn.2001-05.com.
equallogic:0-8a0906-19f91850c-067000000b4532cl-volume-74cf9c04-4543-47ae-
a937-a9b7c6c921e7
ActualMembers: 1
Snap-Warn: 10%
Snap-Depletion: delete-oldest
Description:
Snap-Reserve: 100%
Snap-Reserve-Avail: 100% (1.01GB)
Permission: read-write
DesiredStatus: online
Status: online
Connections: O
Snapshots: O
Bind:
Type: not-replicated
ReplicationReserveSpace: 0MB
```

3. Create a new image from this volume:

```
$ cinder upload-to-image --disk-format raw \
  --container-format bare volume1 image_from_volume1
+---------------------+---------------------------------------+
|       Property      |                 Value                 |
+---------------------+---------------------------------------+
|   container_format  |                  bare                 |
|     disk_format     |                  raw                  |
| display_description |                  None                 |
|          id         | 74cf9c04-4543-47ae-a937-a9b7c6c921e7  |
|       image_id      | 3020a21d-ba37-4495-8899-07fc201161b9  |
|      image_name     |          image_from_volume1           |
|         size        |                   1                   |
|        status       |               uploading               |
|      updated_at     |      2014-03-21T18:31:55.000000       |
|     volume_type     |                  None                 |
+---------------------+---------------------------------------+
```

4. When you uploaded the volume in the previous step, the Image service reported the volume's size as `1` (GB). However, when using **glance image-list** to list the image, the displayed size is 1085276160 bytes, or roughly 1.01 GB:

Table 6.1. Image settings reported by glance image-list for image ID

Name	Disk Format	Container Format	Size	Status
image_from_volume	raw	bare	*1085276160*	active

5. Create a new volume using the previous image (`image_id 3020a21d-ba37-4495-8899-07fc201161b9` in this example) as the source. Set the target volume size to 1 GB; this is the size reported by the **cinder** tool when you uploaded the volume to the Image service:

   ```
   $ cinder create --display-name volume2 \
     --image-id 3020a21d-ba37-4495-8899-07fc201161b9 1
   ERROR: Invalid input received: Size of specified image 2 is larger
   than volume size 1. (HTTP 400) (Request-ID: req-4b9369c0-dec5-4e16-a114-
   c0cdl6bSd210)
   ```

   The attempt to create a new volume based on the size reported by the **cinder** tool will then fail.

Solution

To work around this problem, increase the target size of the new image to the next whole number. In the problem example, you created a 1 GB volume to be used as volume-backed image, so a new volume using this volume-backed image should use a size of 2 GB:

```
$ cinder create --display-name volume2 \
  --image-id 3020a21d-ba37-4495-8899-07fc201161b9 1
+---------------------+--------------------------------------+
|       Property      |                Value                 |
+---------------------+--------------------------------------+
|    attachments      |                  []                  |
| availability_zone   |                 nova                 |
|      bootable       |                false                 |
|     created_at      |      2014-03-21T19:25:31.564482      |
| display_description |                 None                 |
|     display_name    |               volume2                |
|          id         | 64e8eb18-d23f-437b-bcac-b3S2afa6843a |
|    image_id         | 3020a21d-ba37-4495-8899-07fc20116lb9 |
|    metadata         |                  []                  |
|      size           |                  2                   |
| snapshot_id         |                 None                 |
| source_volid        |                 None                 |
|    status           |               creating               |
| volume_type         |                 None                 |
+---------------------+--------------------------------------+
```

Note

The dashboard suggests a suitable size when you create a new volume based on a volume-backed image.

You can then check this new volume into the EQL array:

```
eql> volume select volume-64e8eb18-d23f-437b-bcac-b352afa6843a
eql (volume_volume-61e8eb18-d23f-437b-bcac-b352afa6843a)> show
______________________________ Volume Information
 _______________________________
Name: volume-64e8eb18-d23f-437b-bcac-b352afa6843a
Size: 2GB
VolReserve: 2.01GB
VolReserveInUse: 1.01GB
ReplReserveInUse: 0MB
iSCSI Alias: volume-64e8eb18-d23f-437b-bcac-b352afa6843a
iSCSI Name: iqn.2001-05.com.equallogic:0-8a0906-e3091850e-eae000000b7S32cl-
volume-64e8eb18-d23f-437b-bcac-b3S2afa6Bl3a
ActualMembers: 1
Snap-Warn: 10%
Snap-Depletion: delete-oldest
Description:
Snap-Reserve: 100%
Snap-Reserve-Avail: 100% (2GB)
Permission: read-write
DesiredStatus: online
Status: online
Connections: 1
Snapshots: 0
Bind:
Type: not-replicated
ReplicationReserveSpace: 0MB
```

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.

To configure rich network topologies, you can create and configure networks and subnets and instruct 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 enables 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 which to configure the socket.

Load-Balancer-as-a-Service (LBaaS) overview

Load-Balancer-as-a-Service (LBaaS) enables Networking to distribute incoming requests evenly among designated instances. This distribution ensures that the workload is shared predictably among instances and enables more effective use of system resources. Use one of these load balancing methods to distribute incoming requests:

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.

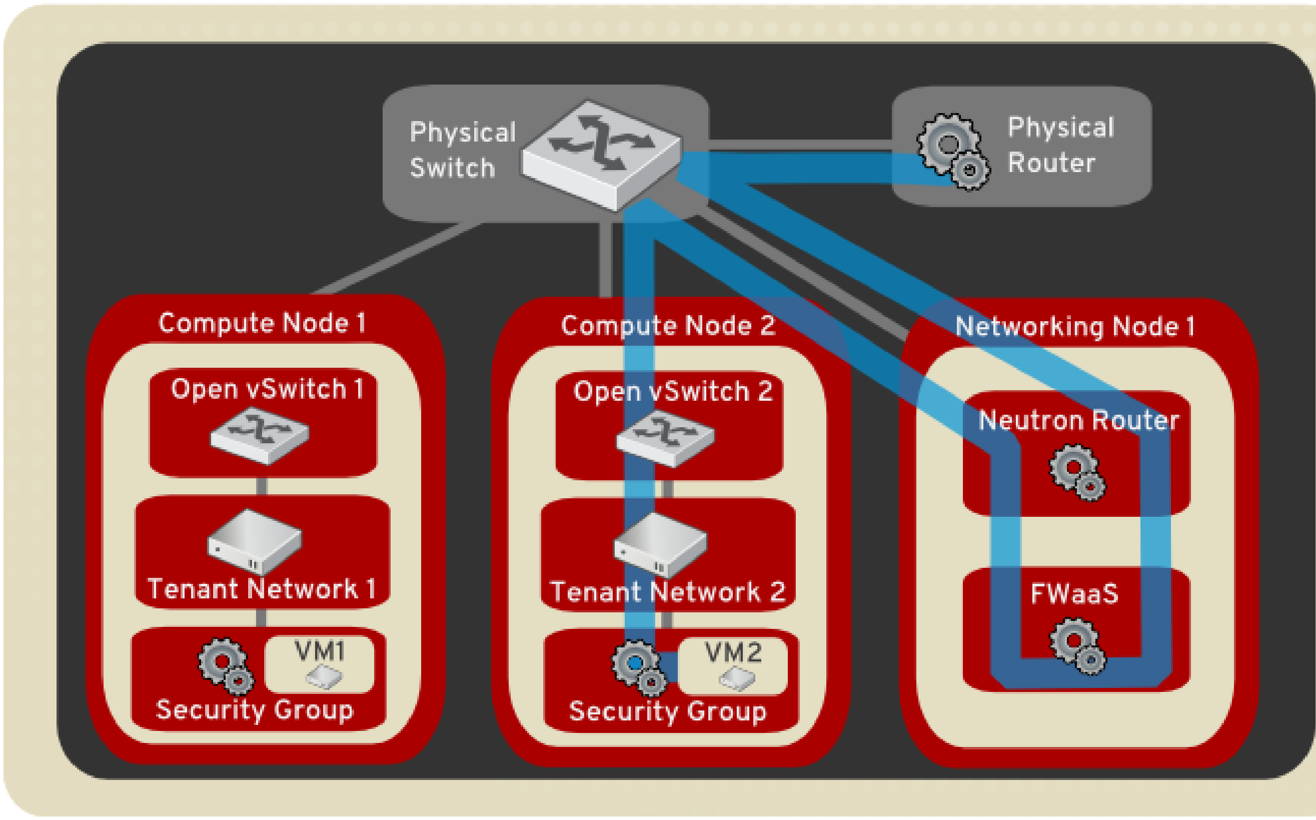
Whereas security groups operate at the instance-level, FWaaS operates at the perimeter to filter traffic at the neutron router.

Note

FWaaS is currently in technical preview; untested operation is not recommended.

The example diagram illustrates the flow of ingress and egress traffic for the VM2 instance:

Figure 7.1. FWaaS architecture



To enable FWaaS

FWaaS management options are also available in the OpenStack dashboard.

1. Enable the FWaaS plug-in in the `/etc/neutron/neutron.conf` file:

```
service_plugins = firewall
[service_providers]
...
service_provider = FIREWALL:Iptables:neutron.agent.linux.
iptables_firewall.OVSHybridIptablesFirewallDriver:default

[fwaas]
driver = neutron_fwaas.services.firewall.drivers.linux.iptables_fwaas.
IptablesFwaasDriver
enabled = True
```

Note

On Ubuntu, modify the `[fwaas]` section in the `/etc/neutron/fwaas_driver.ini` file instead of `/etc/neutron/neutron.conf`.

2. Create the required tables in the database:

```
# neutron-db-manage --service fwaas upgrade head
```

3. Enable the option in the `/usr/share/openstack-dashboard/openstack_dashboard/local/local_settings.py` file, which is typically located on the controller node:

   ```
   OPENSTACK_NEUTRON_NETWORK = {
       ...
       'enable_firewall' = True,
       ...}
   ```

4. Restart the `neutron-l3-agent` and `neutron-server` services to apply the settings.

To configure Firewall-as-a-Service

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 Networking client requires a protocol value; if the rule is protocol agnostic, you can use the `any` value.

2. Create a firewall policy:

   ```
   $ neutron firewall-policy-create --firewall-rules
    "FIREWALL_RULE_IDS_OR_NAMES" myfirewallpolicy
   ```

   Separate firewall rule IDs or names with spaces. The order in which you specify the rules is important.

   You can create a firewall policy without any rules and add rules later, as follows:

   - To add multiple rules, use the update operation.
   - To add a single rule, use the insert-rule operation.

   For more details, see Networking command-line client in the *OpenStack Command-Line Interface Reference*.

   ### 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 you create a Networking router and attach an interface to it.

Allowed-address-pairs. `Allowed-address-pairs` enable 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

Currently, only the ML2, Open vSwitch, and VMware NSX plug-ins support the allowed-address-pairs extension.

Basic allowed-address-pairs operations.

- Create a port with a specified 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 by 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

In releases earlier than Juno, OpenStack Networking prevents setting an allowed address pair on a port that matches the MAC address and one of the fixed IP addresses of the port.

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

To use the REST proxy plug-in with OpenStack Networking

1. Edit the `/etc/neutron/neutron.conf` file and add this line:

```
core_plugin = bigswitch
```

2. In the `/etc/neutron/neutron.conf` file, set the `service_plugins` option:

```
service_plugins = neutron.plugins.bigswitch.l3_router_plugin.L3RestProxy
```

3. Edit the `/etc/neutron/plugins/bigswitch/restproxy.ini` file for the plug-in 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.)

4. Restart `neutron-server` to apply the settings:

   ```
   # service neutron-server restart
   ```

Configure Brocade plug-in

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://github.com/brocade/ncclient
   ```

   As `root`, run this command:

   ```
   # cd ncclient;python setup.py install
   ```

2. Edit the `/etc/neutron/neutron.conf` file and set the following option:

   ```
   core_plugin = brocade
   ```

3. Edit the `/etc/neutron/plugins/brocade/brocade.ini` 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 settings:

   ```
   # service neutron-server restart
   ```

Configure NSX-mh plug-in

Configuring OpenStack Networking to use the NSX multi hypervisor plug-in

The instructions in this section refer to the VMware NSX-mh platform, formerly known as Nicira NVP.

1. Install the NSX plug-in:

   ```
   # apt-get install neutron-plugin-vmware
   ```

2. Edit the `/etc/neutron/neutron.conf` file and set this line:

   ```
   core_plugin = vmware
   ```

   Example `neutron.conf` file for NSX-mh integration:

```
core_plugin = vmware
rabbit_host = 192.168.203.10
allow_overlapping_ips = True
```

3. To configure the NSX-mh controller cluster for OpenStack Networking, locate the `[default]` section in the `/etc/neutron/plugins/vmware/nsx.ini` file and add the following entries:

   - To establish and configure the connection with the controller cluster you must set some parameters, including NSX-mh API endpoints, access credentials, and optionally specify settings for HTTP timeouts, redirects and retries in case of connection failures:

     ```
     nsx_user = ADMIN_USER_NAME
     nsx_password = NSX_USER_PASSWORD
     http_timeout = HTTP_REQUEST_TIMEOUT # (seconds) default 75 seconds
     retries = HTTP_REQUEST_RETRIES # default 2
     redirects = HTTP_REQUEST_MAX_REDIRECTS # default 2
     nsx_controllers = API_ENDPOINT_LIST # comma-separated list
     ```

     To ensure correct operations, the `nsx_user` user must have administrator credentials on the NSX-mh 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-mh plug-in does not perform this check, and results might be unpredictable.

     When you specify multiple API endpoints, the plug-in takes care of load balancing requests on the various API endpoints.

   - The UUID of the NSX-mh transport zone that should be used by default when a tenant creates a network. You can get this value from the **Transport Zones** page for the NSX-mh manager:

     Alternatively the transport zone identifier can be retrieved by query the NSX-mh API: `/ws.v1/transport-zone`

     ```
     default_tz_uuid = TRANSPORT_ZONE_UUID
     ```

   -
     ```
     default_l3_gw_service_uuid = GATEWAY_SERVICE_UUID
     ```

     Warning

     Ubuntu packaging currently does not update the neutron init script to point to the NSX-mh configuration file. Instead, you must manually update `/etc/default/neutron-server` to add this line:

     ```
     NEUTRON_PLUGIN_CONFIG = /etc/neutron/plugins/vmware/nsx.ini
     ```

   For database configuration, see Install Networking Services in the *Installation Guide*.

4. Restart `neutron-server` to apply settings:

```
# service neutron-server restart
```

Warning

The neutron NSX-mh plug-in does not implement initial re-synchronization of Neutron resources. Therefore resources that might already exist in the database when Neutron is switched to the NSX-mh plug-in will not be created on the NSX-mh backend upon 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-mh controllers and the SQL server, and whether all UUID values are correct.

Configure PLUMgrid plug-in

To use the PLUMgrid plug-in with OpenStack Networking

1. Edit the `/etc/neutron/neutron.conf` file and set this line:

   ```
   core_plugin = plumgrid
   ```

2. Edit the `[PLUMgridDirector]` section in the `/etc/neutron/plugins/plumgrid/plumgrid.ini` file 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 the `neutron-server` service to apply the 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: 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 run 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.

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.

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.

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" [212].

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, you must manually run this command or write your own system script that runs on boot before the `neutron-dhcp-agent` service starts.

Networking dhcp-agent can use dnsmasq driver which supports stateful and stateless DHCPv6 for subnets created with `--ipv6_address_mode` set to `dhcpv6-stateful` or `dhcpv6-stateless`.

For example:

```
$ neutron subnet-create --ip-version 6 --ipv6_ra_mode dhcpv6-stateful --
ipv6_address_mode dhcpv6-stateful NETWORK CIDR
```

```
$ neutron subnet-create --ip-version 6 --ipv6_ra_mode dhcpv6-stateless --
ipv6_address_mode dhcpv6-stateless NETWORK CIDR
```

If no dnsmasq process for subnet's network is launched, Networking will launch a new one on subnet's dhcp port in `qdhcp-XXX` namespace. If previous dnsmasq process is already launched, restart dnsmasq with a new configuration.

Networking will update dnsmasq process and restart it when subnet gets updated.

Note

For dhcp-agent to operate in IPv6 mode use at least dnsmasq v2.63.

After a certain, configured timeframe, networks uncouple from DHCP agents when the agents are no longer in use. You can configure the DHCP agent to automatically detach from a network when the agent is out of service, or no longer needed.

This feature applies to all plug-ins that support DHCP scaling. For more information, see the DHCP agent configuration options listed in the OpenStack Configuration Reference.

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
```

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:

- 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.

- IBM SDN-VE plug-in
- MidoNet plug-in
- NSX plug-in
- PLUMgrid plug-in

Warning

Do not configure or use `neutron-l3-agent` if you use one of these plug-ins.

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-*ROUTER_UUID*. 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
   ```

Note

For iproute version 3.12.0 and above, networking namespaces are configured to be deleted by default. This behavior can be changed for both DHCP and L3 agents. The configuration files are `/etc/neutron/dhcp_agent.ini` and `/etc/neutron/l3_agent.ini` respectively.

For DHCP namespace the configuration key: `dhcp_delete_namespaces = True`. You can set it to *False* in case namespaces cannot be deleted cleanly on the host running the DHCP agent.

For L3 namespace, the configuration key: `router_delete_namespaces = True`. You can set it to *False* in case namespaces cannot be deleted cleanly on the host running the L3 agent.

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, 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

The Neutron Metering agent resides beside `neutron-l3-agent`.

To install the metering agent and configure the node

1. Install the agent by running:

   ```
   # apt-get install neutron-metering-agent
   ```

2. If you use one of the following plug-ins, you need to configure the metering agent with these lines as well:

   - An OVS-based plug-in such as OVS, NSX, 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 the `service_plugins` option in the `/etc/neutron/neutron.conf` file on the host that runs `neutron-server`:

   ```
   service_plugins = metering
   ```

   If this option is already defined, add `metering` to the list, using a comma as separator. For example:

   ```
   service_plugins = router,metering
   ```

Configure Load-Balancer-as-a-Service (LBaaS)

Configure Load-Balancer-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, and NSX.

To configure LBaas with Open vSwitch or Linux Bridge plug-in

1. Install the agent:

   ```
   # apt-get install neutron-lbaas-agent haproxy
   ```

2. Enable the HAProxy plug-in by using the `service_provider` option in the `/etc/neutron/neutron.conf` file:

   ```
   service_provider = LOADBALANCER:Haproxy:neutron_lbaas.
   services.loadbalancer.drivers.haproxy.plugin_driver.
   HaproxyOnHostPluginDriver:default
   ```

Warning

The `service_provider` option is already defined in the `/usr/share/neutron/neutron-dist.conf` file on Red Hat based systems. Do not define it in `neutron.conf` otherwise the Networking services will fail to restart.

3. Enable the load-balancing plug-in by using the `service_plugins` option in the `/etc/neutron/neutron.conf` file:

   ```
   service_plugins = lbaas
   ```

   If this option is already defined, add `lbaas` to the list, using a comma as separator. For example:

   ```
   service_plugins = router,lbaas
   ```

4. Enable the HAProxy load balancer in the `/etc/neutron/lbaas_agent.ini` file:

   ```
   device_driver = neutron_lbaas.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
   ```

6. Create the required tables in the database:

   ```
   # neutron-db-manage --service lbaas upgrade head
   ```

7. Apply the settings by restarting the `neutron-server` and `neutron-lbaas-agent` services.

8. Enable load balancing in the **Project** section of the dashboard.

   Change the `enable_lb` option to `True` in the `local_settings` file (on Fedora, RHEL, and CentOS: `/etc/openstack-dashboard/local_settings`, on Ubuntu and Debian: `/etc/openstack-dashboard/local_settings.py`, and on openSUSE and SLES: `/srv/www/openstack-dashboard/openstack_dashboard/local/local_settings.py`):

   ```
   OPENSTACK_NEUTRON_NETWORK = {
       'enable_lb': True,
       ...
   }
   ```

   Apply the settings by restarting the web server. You can now view the Load Balancer management options in the **Project** view in the dashboard.

Configure Hyper-V L2 agent

Before you install the OpenStack Networking Hyper-V L2 agent on a Hyper-V compute node, ensure the compute node has been configured correctly using these instructions.

To install the OpenStack Networking Hyper-V agent and configure the node

1. Download the OpenStack Networking code from the repository:

```
> cd C:\OpenStack\
> git clone https://git.openstack.org/cgit/openstack/neutron
```

2. Install the OpenStack Networking Hyper-V Agent:

```
> cd C:\OpenStack\neutron\
> python setup.py install
```

3. Copy the `policy.json` file:

```
> xcopy C:\OpenStack\neutron\etc\policy.json C:\etc\
```

4. Create the `C:\etc\neutron-hyperv-agent.conf` file and add the proper configuration options and the Hyper-V related options. Here is a sample config file:

```
[DEFAULT]
verbose = true
control_exchange = neutron
policy_file = C:\etc\policy.json
rpc_backend = neutron.openstack.common.rpc.impl_kombu
rabbit_host = IP_ADDRESS
rabbit_port = 5672
rabbit_userid = guest
rabbit_password = <password>
logdir = C:\OpenStack\Log
logfile = neutron-hyperv-agent.log

[AGENT]
polling_interval = 2
physical_network_vswitch_mappings = *:YOUR_BRIDGE_NAME
enable_metrics_collection = true

[SECURITYGROUP]
firewall_driver = neutron.plugins.hyperv.agent.security_groups_driver.
HyperVSecurityGroupsDriver
enable_security_group = true
```

5. Start the OpenStack Networking Hyper-V agent:

```
> C:\Python27\Scripts\neutron-hyperv-agent.exe --config-file C:\etc\
neutron-hyperv-agent.conf
```

Basic operations on agents

This table shows examples of Networking commands that enable you to complete basic operations on agents:

Table 7.3. Basic operations on Networking agents

Operation	Command
List all available agents.	`$ neutron agent-list`
Show information of a given agent.	`$ neutron agent-show AGENT_ID`
Update the admin status and description for a specified agent. The command can be used to enable and disable agents by using `--admin-state-up` parameter set to `False` or `True`.	`$ neutron agent-update --admin-state-up False AGENT_ID`
Delete a given agent. Consider disabling the agent before deletion.	`$ neutron agent-delete AGENT_ID`

See the *OpenStack Command-Line Interface Reference* for more information on Networking commands.

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 7.4. 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.

VMware NSX integration

OpenStack Networking uses the NSX plug-in to integrate with an existing VMware vCenter deployment. When installed on the network nodes, the NSX plug-in enables a NSX controller to centrally manage configuration settings and push them to managed network nodes. Network nodes are considered managed when they're added as hypervisors to the NSX controller.

The diagrams below depict some VMware NSX deployment examples. The first diagram illustrates the traffic flow between VMs on separate Compute nodes, and the second diagram between two VMs on a single Compute node. Note the placement of the VMware NSX plug-in and the `neutron-server` service on the network node. The green arrow indicates the management relationship between the NSX controller and the network node.

Figure 7.2. VMware NSX deployment example - two Compute nodes

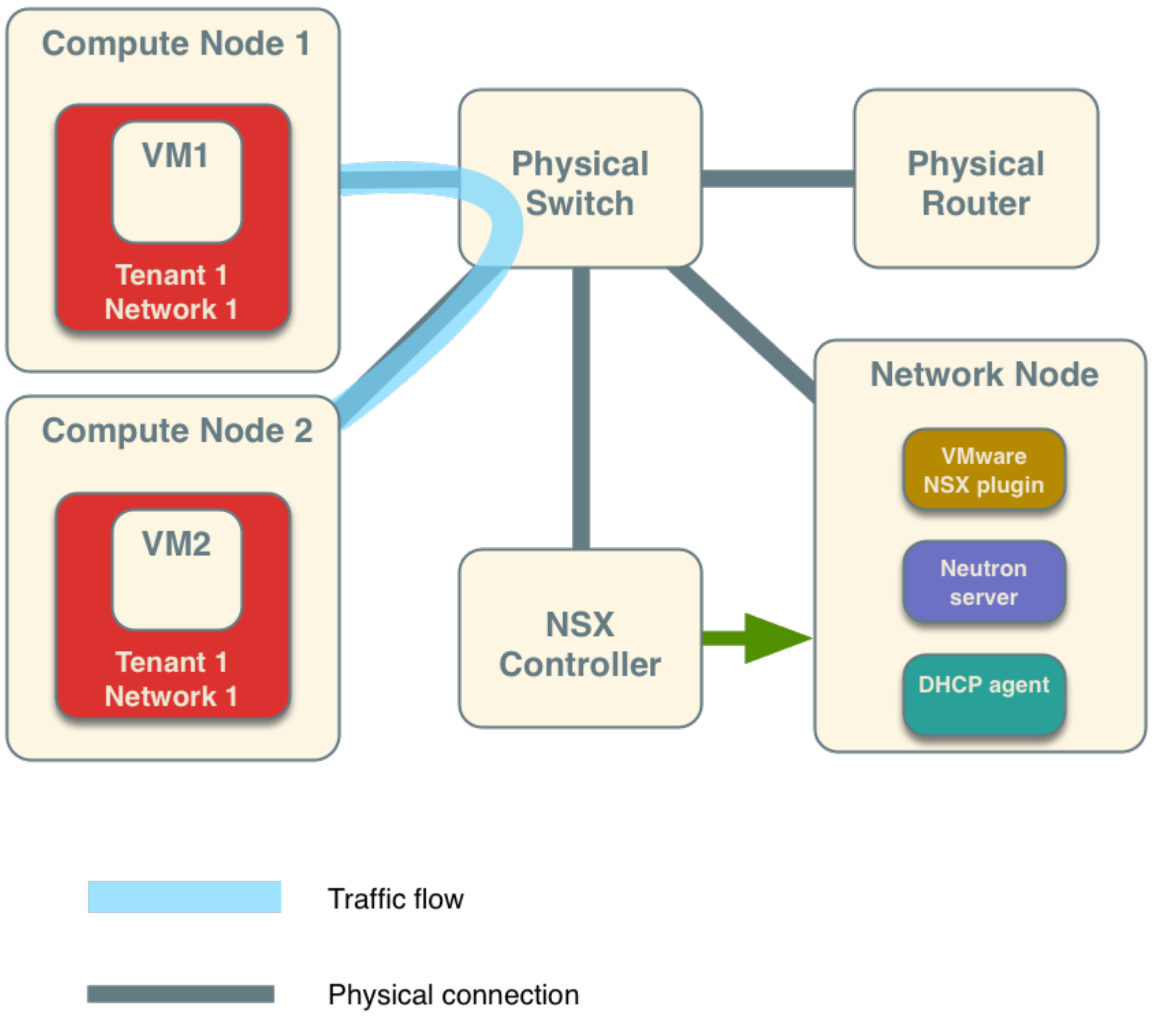
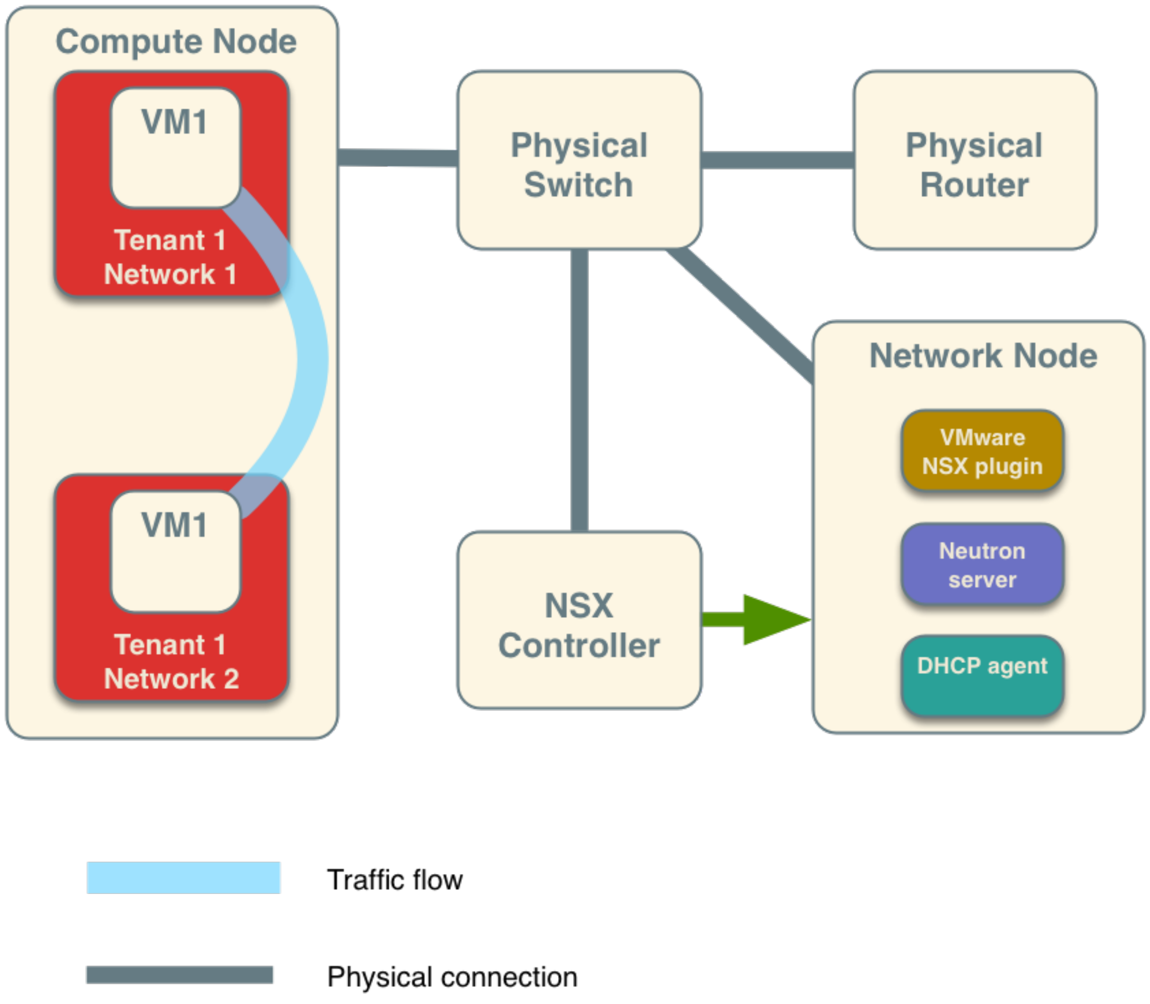


Figure 7.3. VMware NSX deployment example - single Compute node



Configure Identity Service for Networking

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 7.5. nova.conf API and credential settings

Item	Configuration
`[DEFAULT] 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.

Item	Configuration
`[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 7.6. 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 `[neutron]` section in `nova.conf`.

Table 7.7. nova.conf metadata settings

Item	Configuration
`service_metadata_proxy`	Update to `true`, otherwise `nova-api` will not properly respond to requests from the `neutron-metadata-agent`.
`metadata_proxy_shared_secret`	Update to a 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-trust-

Item	Configuration
	ed entities have access to the metadata APIs exposed by `nova-api`.

Note

As a precaution, even when using `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:

```
[DEFAULT]
security_group_api=neutron
network_api_class=nova.network.neutronv2.api.API
firewall_driver=nova.virt.firewall.NoopFirewallDriver

[neutron]
url=http://192.168.1.2:9696
auth_strategy=keystone
admin_tenant_name=service
admin_username=neutron
admin_password=password
admin_auth_url=http://192.168.1.2:35357/v2.0
service_metadata_proxy=true
metadata_proxy_shared_secret=foo
```

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`.

L3 metering agent

You can run an L3 metering agent that enables layer-3 traffic metering. In general, you should launch the metering agent on all nodes that run the L3 agent:

```
neutron-metering-agent --config-file NEUTRON_CONFIG_FILE --config-
file L3_METERING_CONFIG_FILE
```

You must configure a driver that matches the plug-in that runs on the service. The driver adds metering to the routing interface.

Table 7.8. Settings

Option	Value
Open vSwitch	

Option	Value
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 must have the same network namespaces configuration.

Note

If the Linux installation does not support network namespaces, you must disable network namespaces in the L3 metering configuration file. The default value of the `use_namespaces` option is `True`.

```
use_namespaces = False
```

L3 metering driver

You must configure any driver that implements the metering abstraction. Currently the only available implementation uses iptables for metering.

```
driver = neutron.services.metering.drivers.iptables.iptables_driver.
IptablesMeteringDriver
```

L3 metering service driver

To enable L3 metering, you must set the following option in the `neutron.conf` file on the host that runs `neutron-server`:

```
service_plugins = metering
```

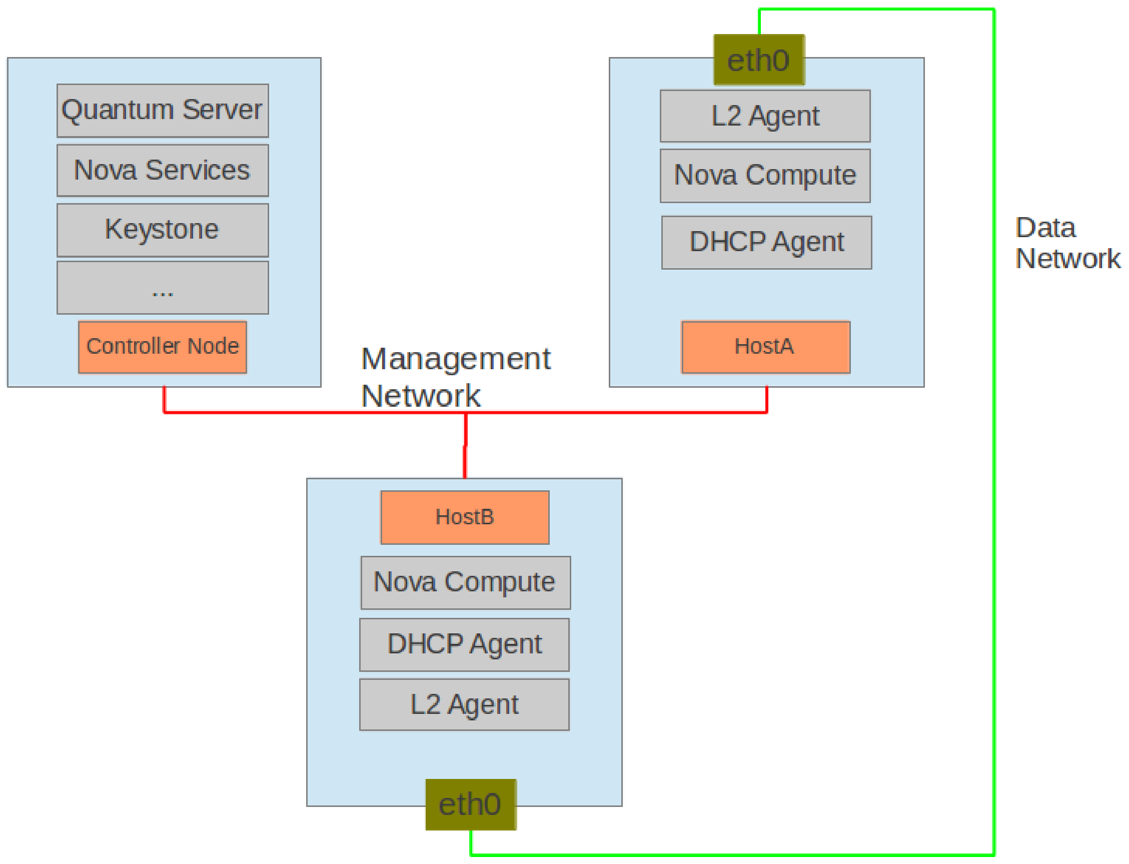
Scalable and highly available DHCP agents

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 7.9. Hosts for demo

Host	Description
OpenStack controller host - controlnode	Runs the Networking, Identity, and Compute 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

controlnode: neutron server

1. Neutron configuration file `/etc/neutron/neutron.conf`:

```
[DEFAULT]
core_plugin = linuxbridge
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
   ```

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
   firewall_driver=nova.virt.firewall.NoopFirewallDriver

   [neutron]
   admin_username=neutron
   admin_password=servicepassword
   admin_auth_url=http://controlnode:35357/v2.0/
   auth_strategy=keystone
   admin_tenant_name=servicetenant
   url=http://100.1.1.10:9696/
   ```

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 the Identity Service.

```
export OS_USERNAME=admin
export OS_PASSWORD=adminpassword
export OS_TENANT_NAME=admin
export OS_AUTH_URL=http://controlnode:5000/v2.0/
```

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 |
+--------------------------------------+------
+--------------------------------------+
```

Manage agents in neutron deployment

Every agent that 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-show** 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.

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.

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           | :-)   |
+--------------------------------------+-------+----------------+-------+
```

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.

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 by 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.

Cloud administrators and tenants can use OpenStack Networking to build rich network topologies. Cloud administrators can create network connectivity on behalf of tenants.

Core Networking API features

After you install and configure Networking, tenants and administrators can perform create-read-update-delete (CRUD) API networking operations by using the Networking API directly or 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, see the *OpenStack End User Guide*.

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 7.10. 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 7.11. 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 8.8.4.4 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 7.12. 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 IMAGE --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 [239].	`$ 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 7.13. Advanced VM creation operations

Operation	Command
Boots a VM with multiple NICs.	`$ nova boot --image IMAGE --flavor FLAVOR --nic net-id=NET1-ID --nic net-id=NET2-ID VM_NAME`
Boots a VM with a specific IP address. Note that you cannot use the `--num-instances` parameter in this case.	`$ nova boot --image IMAGE --flavor FLAVOR --nic net-id=NET-ID,v4-fixed-ip=IP-ADDR 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 IMAGE --flavor FLAVOR VM_NAME`

Note

Cloud images that distribution vendors offer usually have only one active NIC configured. When you boot with multiple NICs, you must configure additional interfaces on the image or the NICS are not reachable.

The following Debian/Ubuntu-based example shows how to set up the interfaces within the instance in the `/etc/network/interfaces` file. You must apply this configuration to the image.

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

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

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 the **neutron security-group-rule-create** command. 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 7.14. 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.

Term	Description
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 7.15. 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`, `gre`, and `vxlan`, corresponding to flat networks, VLAN networks, local networks, GRE networks, and VXLAN networks as defined above. All types of provider networks can be created by administrators, while tenant networks can be implemented as `vlan`, `gre`, `vxlan`, 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.

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.

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 7.16. 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.	`$ neutron router-port-list ROUTER_ID`
	`$ neutron router-port-list ROUTER_NAME`

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 FLOATING_IP_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 FLOATING_IP_ID`
Deletes the floating IP address.	`$ neutron floatingip-delete FLOATING_IP_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 behavior.

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, 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.

Basic security group operations

This table shows example neutron commands that enable you to complete basic security group operations:

Table 7.17. 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`

Operation	Command
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 reference implementation 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` parameter 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  WEBSERVER1_IP --protocol-port 80
   mypool
  $ neutron lb-member-create --address  WEBSERVER2_IP --protocol-port 80
   mypool
  ```

- Creates a health monitor that 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
  ```

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.

Basic VMware NSX QoS operations

This table shows example neutron commands that enable you to complete basic queue operations:

Table 7.18. 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 7.19. 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 an 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 7.20. 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`	10 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`	1 second	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 Networking API.

Table 7.21. 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.

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 7.22. Basic L3 operations

Operation	Command
Creates a metering label.	`$ neutron meter-label-create LABEL1 --description "DESCRIPTION_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`
	For example:
	`$ 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`

Lists the value of created metering label rules.	`$ ceilometer sample-list -m SNMP_MEASUREMENT`
	For example:
	`$ ceilometer sample-list -m hardware.network.bandwidth.bytes`
	`$ ceilometer sample-list -m hardware.network.incoming.bytes`
	`$ ceilometer sample-list -m hardware.network.outgoing.bytes`
	`$ ceilometer sample-list -m hardware.network.outgoing.errors`

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`:

```
# Driver or drivers to handle sending notifications. (multi
# valued)
#notification_driver=

# AMQP topic used for OpenStack notifications. (list value)
# Deprecated group/name - [rpc_notifier2]/topics
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

# Options defined in oslo.messaging
#

# The default exchange under which topics are scoped. May be
# overridden by an exchange name specified in the
# transport_url option. (string value)
#control_exchange=openstack
```

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.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_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 a 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 also be related to specific API extensions; for instance `extension:provider_network:set` is 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 then the 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": [
        [
            "ru❷le: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": [
        [
            "rule:admin_or_owner"
        ],
        [           ❸
            "rule:shared"
        ]
    ],
    "create_network:shared": [
        [
            "rule:admin_only"
        ]
    ],                        ❹
    "update_network": [
        [
            "rule:admin_or_owner"
        ]
    ],
    "delete_network": [
        [
            "rule:admin_or_owner"
        ]
    ],
    "create_port": [],
    "create_port:mac_address": [
        [
            "rule:admin_or_network_owner"
        ]
    ],
    "create_port:fixed_ips": [
        [                       ❺
            "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, 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"]]
        }
```

8. Telemetry

Table of Contents

The Telemetry module is the metering service in OpenStack.

Introduction

Even in the cloud industry, providers must use a multi-step process for billing. The required steps to bill for usage in a cloud environment are metering, rating, and billing. Because the provider's requirements may be far too specific for a shared solution, rating and billing solutions cannot be designed in a common module that satisfies all. Providing users with measurements on cloud services is required to meet the "measured service" definition of cloud computing.

The Telemetry module was originally designed to support billing systems for OpenStack cloud resources. This project only covers the metering portion of the required processing for billing. This module collects information about the system and stores it in the form of samples in order to provide data about anything that can be billed.

In addition to system measurements, the Telemetry module also captures event notifications triggered when various actions are executed in the OpenStack system. This data is captured as Events and stored alongside metering data.

The list of meters is continuously growing, which makes it possible to use the data collected by Telemetry for different purposes, other than billing. For example, the autoscaling feature in the Orchestration module can be triggered by alarms this module sets and then gets notified within Telemetry.

The sections in this document contain information about the architecture and usage of Telemetry. The first section contains a brief summary about the system architecture used in a typical OpenStack deployment. The second section describes the data collection mechanisms. You can also read about alarming to understand how alarm definitions can be posted to Telemetry and what actions can happen if an alarm is raised. The last section contains a troubleshooting guide, which mentions error situations and possible solutions for the problems.

You can retrieve the collected samples three different ways: with the REST API, with the command line interface, or with the Metering tab on an OpenStack dashboard.

System architecture

The Telemetry module uses an agent-based architecture. Several modules combine their responsibilities to collect data, store samples in a database, or provide an API service for handling incoming requests.

The Telemetry module is built from the following agents and services:

`ceilometer-api`	Presents aggregated metering data to consumers (such as billing engines, analytics tools and so forth).
`ceilometer-polling`	Polls for different kinds of meter data by using the polling plug-ins (pollsters) registered in different namespaces.
`ceilometer-agent-central`	Polls the public RESTful APIs of other OpenStack services such as Compute service and Image service, in order to keep tabs on resource existence, by using the polling plug-ins (pollsters) registered in the central polling namespace.
`ceilometer-agent-compute`	Polls the local hypervisor or libvirt daemon to acquire performance data for the local instances, messages and emits the data as AMQP messages, by using the polling plug-ins (pollsters) registered in the compute polling namespace.
`ceilometer-agent-ipmi`	Polls the local node with IPMI support, in order to acquire IPMI sensor data and Intel Node Manager data, by using the polling plug-ins (pollsters) registered in the IPMI polling namespace.
`ceilometer-agent-notification`	Consumes AMQP messages from other OpenStack services.
`ceilometer-collector`	Consumes AMQP notifications from the agents, then dispatches these data to the appropriate data store.
`ceilometer-alarm-evaluator`	Determines when alarms fire due to the associated statistic trend crossing a threshold over a sliding time window.
`ceilometer-alarm-notifier`	Initiates alarm actions, for example calling out to a webhook with a description of the alarm state transition.

Note

The `ceilometer-polling` service is available since the Kilo release.

Besides the `ceilometer-agent-compute` and the `ceilometer-agent-ipmi` service, all the other services are placed on one or more controller nodes.

The Telemetry architecture highly depends on the AMQP service both for consuming notifications coming from OpenStack services and internal communication.

Supported databases

The other key external component of Telemetry is the database, where events, samples, alarm definitions and alarms are stored.

Note

Multiple database back ends can be configured in order to store events, samples and alarms separately.

The list of supported database back ends:

- ElasticSearch (events only)
- MongoDB
- MySQL
- PostgreSQL
- HBase
- DB2

Supported hypervisors

The Telemetry module collects information about the virtual machines, which requires close connection to the hypervisor that runs on the compute hosts.

The list of supported hypervisors is:

- The following hypervisors are supported via Libvirt:
  - Kernel-based Virtual Machine (KVM)
  - Quick Emulator (QEMU)
  - Linux Containers (LXC)
  - User-mode Linux (UML)

  ### Note

  For details about hypervisor support in libvirt please check the Libvirt API support matrix.
- Hyper-V
- XEN
- VMWare vSphere

Supported networking services

Telemetry is able to retrieve information from OpenStack Networking and external networking services:

- OpenStack Networking:
  - Basic network meters
  - Firewall-as-a-Service (FWaaS) meters
  - Loadbalancer-as-a-Service (LBaaS) meters
  - VPN-as-a-Service (VPNaaS) meters
- SDN controller meters:
  - OpenDaylight
  - OpenContrail

Users, roles and tenants

This module of OpenStack uses OpenStack Identity for authenticating and authorizing users. The required configuration options are listed in the Telemetry section in the *OpenStack Configuration Reference*.

Two roles are used in the system basically, which are the 'admin' and 'non-admin'. The authorization happens before processing each API request. The amount of returned data depends on the role the requestor owns.

The creation of alarm definitions also highly depends on the role of the user, who initiated the action. Further details about alarm handling can be found in the section called “Alarms” [284] in this guide.

Data collection

The main responsibility of Telemetry in OpenStack is to collect information about the system that can be used by billing systems or interpreted by analytic tooling. The original focus, regarding to the collected data, was on the counters that can be used for billing, but the range is getting wider continuously.

Collected data can be stored in the form of samples or events in the supported databases, listed in the section called “Supported databases” [261].

Samples can have various sources regarding to the needs and configuration of Telemetry, which requires multiple methods to collect data.

The available data collection mechanisms are:

Notifications Processing notifications from other OpenStack services, by consuming messages from the configured message queue system.

Polling	Retrieve information directly from the hypervisor or from the host machine using SNMP, or by using the APIs of other OpenStack services.
RESTful API	Pushing samples via the RESTful API of Telemetry.

Notifications

All the services send notifications about the executed operations or system state in OpenStack. Several notifications carry information that can be metered, like the CPU time of a VM instance created by OpenStack Compute service.

The Telemetry module has a separate agent that is responsible for consuming notifications, namely the notification agent. This component is responsible for consuming from the message bus and transforming notifications into events and measurement samples.

The different OpenStack services emit several notifications about the various types of events that happen in the system during normal operation. Not all these notifications are consumed by the Telemetry module, as the intention is only to capture the billable events and notifications that can be used for monitoring or profiling purposes. The notification agent filters by the event type, that is contained by each notification message. The following table contains the event types by each OpenStack service that are transformed to samples by Telemetry.

Table 8.1. Consumed event types from OpenStack services

OpenStack service	Event types	Note
OpenStack Compute	scheduler.run_instance.scheduled scheduler.select_destinations compute.instance.*	For a more detailed list of Compute notifications please check the System Usage Data wiki page.
Bare metal service	hardware.ipmi.*	
OpenStack Image service	image.update image.upload image.delete image.send	The required configuration for Image service can be found in the Configure the Image service for Telemetry section section in the *OpenStack Installation Guide*.
OpenStack Networking	floatingip.create.end floatingip.update.* floatingip.exists network.create.end network.update.* network.exists port.create.end port.update.* port.exists router.create.end router.update.* router.exists	

OpenStack service	Event types	Note
	subnet.create.end subnet.update.* subnet.exists l3.meter	
Orchestration module	orchestration.stack.create.end orchestration.stack.update.end orchestration.stack.delete.end orchestration.stack.resume.end orchestration.stack.suspend.end	
OpenStack Block Storage	volume.exists volume.create.* volume.delete.* volume.update.* volume.resize.* volume.attach.* volume.detach.* snapshot.exists snapshot.create.* snapshot.delete.* snapshot.update.*	The required configuration for Block Storage service can be found in the Add the Block Storage service agent for Telemetry section section in the *OpenStack Installation Guide*.

Note

Some services require additional configuration to emit the notifications using the correct control exchange on the message queue and so forth. These configuration needs are referred in the above table for each OpenStack service that needs it.

Note

When the `store_events` option is set to True in `ceilometer.conf`, the notification agent needs database access in order to work properly.

Middleware for OpenStack Object Storage service

A subset of Object Store statistics requires an additional middleware to be installed behind the proxy of Object Store. This additional component emits notifications containing data-flow-oriented meters, namely the storage.objects.(incoming|outgoing).bytes values. The list of these meters are listed in the section called “OpenStack Object Storage” [297], marked with `notification` as origin.

The instructions on how to install this middleware can be found in Configure the Object Storage service for Telemetry section in the *OpenStack Installation Guide*.

Telemetry middleware

Telemetry provides the capability of counting the HTTP requests and responses for each API endpoint in OpenStack. This is achieved by storing a sample for each event marked as `audit.http.request`, `audit.http.response`, `http.request` or `http.response`.

It is recommended that these notifications be consumed as Events rather than samples to better index the appropriate values and avoid massive load on the Metering database. If preferred, Telemetry can consume these events as samples if the services are configured to emit `http.*` notifications.

Polling

The Telemetry module is intended to store a complex picture of the infrastructure. This goal requires additional information than what is provided by the events and notifications published by each service. Some information is not emitted directly, like resource usage of the VM instances.

Therefore Telemetry uses another method to gather this data by polling the infrastructure including the APIs of the different OpenStack services and other assets, like hypervisors. The latter case requires closer interaction with the compute hosts. To solve this issue, Telemetry uses an agent based architecture to fulfill the requirements against the data collection.

There are three types of agents supporting the polling mechanism, the compute agent, the central agent, and the IPMI agent. Under the hood, all the types of polling agents are the same `ceilometer-polling` agent, except that they load different polling plug-ins (pollsters) from different namespaces to gather data. The following subsections give further information regarding the architectural and configuration details of these components.

Running `ceilometer-agent-compute` is exactly the same as:

```
$ ceilometer-polling --polling-namespaces compute
```

Running `ceilometer-agent-central` is exactly the same as:

```
$ ceilometer-polling --polling-namespaces central
```

Running `ceilometer-agent-ipmi` is exactly the same as:

```
$ ceilometer-polling --polling-namespaces ipmi
```

In addition to loading all the polling plug-ins registered in the specified namespaces, the `ceilometer-polling` agent can also specify the polling plug-ins to be loaded by using the `pollster-list` option:

```
$ ceilometer-polling --polling-namespaces central \
        --pollster-list image image.size storage.*
```

Note

HA deployment is NOT supported if the `pollster-list` option is used.

Note

The `ceilometer-polling` service is available since Kilo release.

Central agent

As the name of this agent shows, it is a central component in the Telemetry architecture. This agent is responsible for polling public REST APIs to retrieve additional information on OpenStack resources not already surfaced via notifications, and also for polling hardware resources over SNMP.

The following services can be polled with this agent:

- OpenStack Networking
- OpenStack Object Storage
- OpenStack Block Storage
- Hardware resources via SNMP
- Energy consumption meters via Kwapi framework

To install and configure this service use the Install the Telemetry module section in the *OpenStack Installation Guide*.

The central agent does not need direct database connection. The samples collected by this agent are sent via AMQP to the collector service or any external service, which is responsible for persisting the data into the configured database back end.

Compute agent

This agent is responsible for collecting resource usage data of VM instances on individual compute nodes within an OpenStack deployment. This mechanism requires a closer interaction with the hypervisor, therefore a separate agent type fulfills the collection of the related meters, which is placed on the host machines to locally retrieve this information.

A compute agent instance has to be installed on each and every compute node, installation instructions can be found in the Install the Compute agent for Telemetry section in the *OpenStack Installation Guide*.

Just like the central agent, this component also does not need a direct database connection. The samples are sent via AMQP to the collector.

The list of supported hypervisors can be found in the section called “Supported hypervisors” [261]. The compute agent uses the API of the hypervisor installed on the compute hosts. Therefore the supported meters may be different in case of each virtualization back end, as each inspection tool provides a different set of meters.

The list of collected meters can be found in the section called “OpenStack Compute” [289]. The support column provides the information that which meter is available for each hypervisor supported by the Telemetry module.

Note

Telemetry supports Libvirt, which hides the hypervisor under it.

Support for HA deployment of the central and compute agent services

Both the central and the compute agent can run in an HA deployment, which means that multiple instances of these services can run in parallel with workload partitioning among these running instances.

The Tooz library provides the coordination within the groups of service instances. It provides an API above several back ends that can be used for building distributed applications.

Tooz supports various drivers including the following back end solutions:

- Zookeeper. Recommended solution by the Tooz project.
- Redis. Recommended solution by the Tooz project.
- Memcached Recommended for testing.

You must configure a supported Tooz driver for the HA deployment of the Telemetry services.

For information about the required configuration options that have to be set in the `ceilometer.conf` configuration file for both the central and compute agents, see the `coordination` section in the *OpenStack Configuration Reference*.

Note

Without the `backend_url` option being set only one instance of both the central and compute agent service is able to run and function correctly.

The availability check of the instances is provided by heartbeat messages. When the connection with an instance is lost, the workload will be reassigned within the remained instances in the next polling cycle.

Note

`Memcached` uses a `timeout` value, which should always be set to a value that is higher than the `heartbeat` value set for Telemetry.

For backward compatibility and supporting existing deployments, the central agent configuration also supports using different configuration files for groups of service instances of this type that are running in parallel. For enabling this configuration set a value for the `partitioning_group_prefix` option in the `central` section in the *OpenStack Configuration Reference*.

Warning

For each sub-group of the central agent pool with the same `partitioning_group_prefix` a disjoint subset of meters must be polled, otherwise samples may be missing or duplicated. The list of meters to poll can be set in the `/etc/ceilometer/pipeline.yaml` configuration file. For more information about pipelines see the section called “Data collection and processing” [269].

To enable the compute agent to run multiple instances simultaneously with workload partitioning, the `workload_partitioning` option has to be set to `True` under the compute section in the `ceilometer.conf` configuration file.

IPMI agent

This agent is responsible for collecting IPMI sensor data and Intel Node Manager data on individual compute nodes within an OpenStack deployment. This agent requires an IPMI capable node with ipmitool installed, which is a common utility for IPMI control on various Linux distributions.

An IPMI agent instance could be installed on each and every compute node with IPMI support, except when the node is managed by the Bare metal service and the `conductor.send_sensor_data` option is set to `true` in the Bare metal service. It is no harm to install this agent on a compute node without IPMI or Intel Node Manager support, as the agent checks for the hardware and if none is available, returns empty data. It is suggested that you install the IPMI agent only on an IPMI capable node for performance reasons.

Just like the central agent, this component also does not need direct database access. The samples are sent via AMQP to the collector.

The list of collected meters can be found in the section called “Bare metal service” [294].

Note

Do not deploy both the IPMI agent and the Bare metal service on one compute node. If `conductor.send_sensor_data` set, this misconfiguration causes duplicated IPMI sensor samples.

Send samples to Telemetry

While most parts of the data collection in the Telemetry module are automated, Telemetry provides the possibility to submit samples via the REST API to allow users to send custom samples into this module.

This option makes it possible to send any kind of samples without the need of writing extra code lines or making configuration changes.

The samples that can be sent to Telemetry are not limited to the actual existing meters. There is a possibility to provide data for any new, customer defined counter by filling out all the required fields of the POST request.

If the sample corresponds to an existing meter, then the fields like `meter-type` and meter name should be matched accordingly.

The required fields for sending a sample using the command line client are:

- ID of the corresponding resource. (`--resource-id`)
- Name of meter. (`--meter-name`)

- Type of meter. (`--meter-type`)

  Predefined meter types:

  - Gauge
  - Delta
  - Cumulative

- Unit of meter. (`--meter-unit`)
- Volume of sample. (`--sample-volume`)

To send samples to Telemetry using the command line client, the following command should be invoked:

```
$ ceilometer sample-create -r 37128ad6-daaa-4d22-9509-b7e1c6b08697 \
  -m memory.usage --meter-type gauge --meter-unit MB --sample-volume 48
+-------------------+--------------------------------------------+
| Property          | Value                                      |
+-------------------+--------------------------------------------+
| message_id        | 6118820c-2137-11e4-a429-08002715c7fb       |
| name              | memory.usage                               |
| project_id        | e34eaa91d52a4402b4cb8bc9bbd308c1           |
| resource_id       | 37128ad6-daaa-4d22-9509-b7e1c6b08697       |
| resource_metadata | {}                                         |
| source            | e34eaa91d52a4402b4cb8bc9bbd308c1:openstack |
| timestamp         | 2014-08-11T09:10:46.358926                 |
| type              | gauge                                      |
| unit              | MB                                         |
| user_id           | 679b0499e7a34ccb9d90b64208401f8e           |
| volume            | 48.0                                       |
+-------------------+--------------------------------------------+
```

Data collection and processing

The mechanism by which data is collected and processed is called a pipeline. Pipelines, at the configuration level, describe a coupling between sources of data and the corresponding sinks for transformation and publication of data.

A source is a producer of data: samples or events. In effect, it is a set of pollsters or notification handlers emitting datapoints for a set of matching meters and event types.

Each source configuration encapsulates name matching, polling interval determination, optional resource enumeration or discovery, and mapping to one or more sinks for publication.

Data gathered can be used for different purposes, which can impact how frequently it needs to be published. Typically, a meter published for billing purposes needs to be updated every 30 minutes while the same meter may be needed for performance tuning every minute.

Warning

Rapid polling cadences should be avoided, as it results in a huge amount of data in a short time frame, which may negatively affect the performance of both Telemetry and the underlying database back end. We therefore strongly recommend you do not use small granularity values like 10 seconds.

A sink, on the other hand, is a consumer of data, providing logic for the transformation and publication of data emitted from related sources.

In effect, a sink describes a chain of handlers. The chain starts with zero or more transformers and ends with one or more publishers. The first transformer in the chain is passed data from the corresponding source, takes some action such as deriving rate of change, performing unit conversion, or aggregating, before passing the modified data to the next step that is described in the section called “Publishers” [282].

Pipeline configuration

Pipeline configuration by default, is stored in separate configuration files, called `pipeline.yaml` and `event_pipeline.yaml`, next to the `ceilometer.conf` file. The meter pipeline and event pipeline configuration files can be set by the `pipeline_cfg_file` and `event_pipeline_cfg_file` options listed in the Description of configuration options for api table section in the *OpenStack Configuration Reference* respectively. Multiple pipelines can be defined in one pipeline configuration file.

The meter pipeline definition looks like the following:

```
---
sources:
  - name: 'source name'
    interval: 'how often should the samples be injected into the pipeline'
    meters:
      - 'meter filter'
    resources:
      - 'list of resource URLs'
    sinks
      - 'sink name'
sinks:
  - name: 'sink name'
    transformers: 'definition of transformers'
    publishers:
      - 'list of publishers'
```

The interval parameter in the sources section should be defined in seconds. It determines the polling cadence of sample injection into the pipeline, where samples are produced under the direct control of an agent.

There are several ways to define the list of meters for a pipeline source. The list of valid meters can be found in the section called “Measurements” [288]. There is a possibility to define all the meters, or just included or excluded meters, with which a source should operate:

- To include all meters, use the `*` wildcard symbol. It is highly advisable to select only the meters that you intend on using to avoid flooding the metering database with unused data.
- To define the list of meters, use either of the following:
  - To define the list of included meters, use the `meter_name` syntax.
  - To define the list of excluded meters, use the `!meter_name` syntax.
  - For meters, which have variants identified by a complex name field, use the wildcard symbol to select all, e.g. for "instance:m1.tiny", use "instance:*".

Note

Please be aware that we do not have any duplication check between pipelines and if you add a meter to multiple pipelines then it is assumed the duplication is intentional and may be stored multiple times according to the specified sinks.

The above definition methods can be used in the following combinations:

- Use only the wildcard symbol.
- Use the list of included meters.
- Use the list of excluded meters.
- Use wildcard symbol with the list of excluded meters.

Note

At least one of the above variations should be included in the meters section. Included and excluded meters cannot co-exist in the same pipeline. Wildcard and included meters cannot co-exist in the same pipeline definition section.

The optional resources section of a pipeline source allows a static list of resource URLs to be configured for polling.

The transformers section of a pipeline sink provides the possibility to add a list of transformer definitions. The available transformers are:

Table 8.2. List of available transformers

Name of transformer	Reference name for configuration
Accumulator	accumulator
Aggregator	aggregator
Arithmetic	arithmetic
Rate of change	rate_of_change
Unit conversion	unit_conversion

The publishers section contains the list of publishers, where the samples data should be sent after the possible transformations.

Similarly, the event pipeline definition looks like the following:

```
---
sources:
  - name: 'source name'
    events:
      - 'event filter'
    sinks
      - 'sink name'
sinks:
  - name: 'sink name'
    publishers:
      - 'list of publishers'
```

The event filter uses the same filtering logic as the meter pipeline.

Transformers

The definition of transformers can contain the following fields:

name	Name of the transformer.
parameters	Parameters of the transformer.

The parameters section can contain transformer specific fields, like source and target fields with different subfields in case of the rate of change, which depends on the implementation of the transformer.

Rate of change transformer

In the case of the transformer that creates the `cpu_util` meter, the definition looks like the following:

```
transformers:
    - name: "rate_of_change"
      parameters:
          target:
              name: "cpu_util"
              unit: "%"
              type: "gauge"
              scale: "100.0 / (10**9 * (resource_metadata.cpu_number or 1))"
```

The rate of change the transformer generates is the `cpu_util`meter from the sample values of the `cpu` counter, which represents cumulative CPU time in nanoseconds. The transformer definition above defines a scale factor (for nanoseconds and multiple CPUs), which is applied before the transformation derives a sequence of gauge samples with unit '%', from sequential values of the `cpu` meter.

The definition for the disk I/O rate, which is also generated by the rate of change transformer:

```
transformers:
    - name: "rate_of_change"
      parameters:
          source:
              map_from:
                  name: "disk\\.(read|write)\\.(bytes|requests)"
                  unit: "(B|request)"
          target:
              map_to:
                  name: "disk.\\1.\\2.rate"
                  unit: "\\1/s"
              type: "gauge"
```

Unit conversion transformer

Transformer to apply a unit conversion. It takes the volume of the meter and multiplies it with the given 'scale' expression. Also supports `map_from` and `map_to` like the rate of change transformer.

Sample configuration:

```
transformers:
    - name: "unit_conversion"
      parameters:
          target:
              name: "disk.kilobytes"
              unit: "KB"
              scale: "1.0 / 1024.0"
```

With the `map_from` and `map_to` :

```
transformers:
    - name: "unit_conversion"
      parameters:
          source:
              map_from:
                  name: "disk\\.(read|write)\\.bytes"
          target:
              map_to:
                  name: "disk.\\1.kilobytes"
              scale: "1.0 / 1024.0"
              unit: "KB"
```

Aggregator transformer

A transformer that sums up the incoming samples until enough samples have come in or a timeout has been reached.

Timeout can be specified with the `retention_time` option. If we want to flush the aggregation after a set number of samples have been aggregated, we can specify the size parameter.

The volume of the created sample is the sum of the volumes of samples that came into the transformer. Samples can be aggregated by the attributes `project_id` , `user_id` and `resource_metadata`. To aggregate by the chosen attributes, specify them in the configuration and set which value of the attribute to take for the new sample (first to take the first sample's attribute, last to take the last sample's attribute, and drop to discard the attribute).

To aggregate 60s worth of samples by `resource_metadata` and keep the `resource_metadata` of the latest received sample:

```
transformers:
    - name: "aggregator"
      parameters:
          retention_time: 60
          resource_metadata: last
```

To aggregate each 15 samples by `user_id` and `resource_metadata` and keep the `user_id` of the first received sample and drop the `resource_metadata`:

```
transformers:
    - name: "aggregator"
      parameters:
          size: 15
          user_id: first
          resource_metadata: drop
```

Accumulator transformer

This transformer simply caches the samples until enough samples have arrived and then flushes them all down the pipeline at once.

```
transformers:
    - name: "accumulator"
      parameters:
          size: 15
```

Multi meter arithmetic transformer

This transformer enables us to perform arithmetic calculations over one or more meters and/or their metadata, for example:

```
memory_util = 100 * memory.usage / memory
```

A new sample is created with the properties described in the `target` section of the transformer's configuration. The sample's volume is the result of the provided expression. The calculation is performed on samples from the same resource.

Note

The calculation is limited to meters with the same interval.

Example configuration:

```
transformers:
    - name: "arithmetic"
      parameters:
        target:
          name: "memory_util"
          unit: "%"
          type: "gauge"
          expr: "100 * $(memory.usage) / $(memory)"
```

To demonstrate the use of metadata, here is the implementation of a silly meter that shows average CPU time per core:

```
transformers:
    - name: "arithmetic"
      parameters:
        target:
          name: "avg_cpu_per_core"
          unit: "ns"
          type: "cumulative"
          expr: "$(cpu) / ($(cpu).resource_metadata.cpu_number or 1)"
```

Note

Expression evaluation gracefully handles NaNs and exceptions. In such a case it does not create a new sample but only logs a warning.

Block Storage audit script setup to get notifications

If you want to collect OpenStack Block Storage notification on demand, you can use **cin-der-volume-usage-audit** from OpenStack Block Storage. This script becomes available when you install OpenStack Block Storage, so you can use it without any specific settings and you don't need to authenticate to access the data. To use it, you must run this command in the following format:

```
$ cinder-volume-usage-audit \
  --start_time='YYYY-MM-DD HH:MM:SS' --end_time='YYYY-MM-DD HH:MM:SS' --
send_actions
```

This script outputs what volumes or snapshots were created, deleted, or exists in a given period of time and some information about these volumes or snapshots. Information about the existence and size of volumes and snapshots is store in the Telemetry module. This data is also stored as an event which is the recommended usage as it provides better indexing of data.

Using this script via cron you can get notifications periodically, for example, every 5 minutes.

```
*/5 * * * * /path/to/cinder-volume-usage-audit --send_actions
```

Storing samples

The Telemetry module has a separate service that is responsible for persisting the data that comes from the pollsters or is received as notifications. The data can be stored in a file or a database back end, for which the list of supported databases can be found in the section called “Supported databases” [261]. The data can also be sent to an external data store by using an HTTP dispatcher.

The `ceilometer-collector` service receives the data as messages from the message bus of the configured AMQP service. It sends these datapoints without any modification to the configured target. The service has to run on a host machine from which it has access to the configured dispatcher.

Note

Multiple dispatchers can be configured for Telemetry at one time.

Multiple `ceilometer-collector` process can be run at a time. It is also supported to start multiple worker threads per collector process. The `collector_workers` configuration option has to be modified in the collector section of the `ceilometer.conf` configuration file.

Note

Prior to the Juno release, it is not recommended to use multiple workers per collector process when using PostgreSQL as the database back end.

Database dispatcher

When the database dispatcher is configured as data store, you have the option to set a `time_to_live` option (ttl) for samples. By default the time to live value for samples is set to -1, which means that they are kept in the database forever.

The time to live value is specified in seconds. Each sample has a time stamp, and the `ttl` value indicates that a sample will be deleted from the database when the number of seconds has elapsed since that sample reading was stamped. For example, if the time to live is set to 600, all samples older than 600 seconds will be purged from the database.

Certain databases support native TTL expiration. In cases where this is not possible, a command-line script, which you can use for this purpose is `ceilometer-expirer`. You can run it in a cron job, which helps to keep your database in a consistent state.

The level of support differs in case of the configured back end:

Table 8.3. Time-to-live support for database back ends

Database	TTL value support	Note
MongoDB	Yes	MongoDB has native TTL support for deleting samples that are older than the configured ttl value.
SQL-based back ends	Yes	`ceilometer-expirer` has to be used for deleting samples and its related data from the database.
HBase	No	Telemetry's HBase support does not include native TTL nor `ceilometer-expirer` support.
DB2 NoSQL	No	DB2 NoSQL does not have native TTL nor `ceilometer-expirer` support.

HTTP dispatcher

The Telemetry module supports sending samples to an external HTTP target. The samples are sent without any modification. To set this option as the collector's target, the `dispatcher` has to be changed to `http` in the `ceilometer.conf` configuration file. For the list of options that you need to set, see the see the `dispatcher_http` section in the *OpenStack Configuration Reference*.

File dispatcher

You can store samples in a file by setting the `dispatcher` option in `ceilometer.conf` o `file`. For the list of configuration options, see the `dispatcher_file` section in the *OpenStack Configuration Reference*.

Data retrieval

The Telemetry module offers several mechanisms from which the persisted data can be accessed. As described in the section called “System architecture” [260] and in the section called “Data collection” [262] the collected information can be stored in one or more database back ends, which are hidden by the Telemetry RESTful API.

Note

It is highly recommended not to access directly the database and read or modify any data in it. The API layer hides all the changes in the actual database schema and provides a standard interface to expose the samples, alarms and so forth.

Telemetry v2 API

The Telemetry module provides a RESTful API, from which the collected samples and all the related information can be retrieved, like the list of meters, alarm definitions and so forth.

The Telemetry API URL can be retrieved from the service catalog provided by OpenStack Identity, which is populated during the installation process. The API access needs a valid token and proper permission to retrieve data, as described in the section called "Users, roles and tenants" [262].

Further information about the available API endpoints can be found in the Telemetry API Reference.

Query

The API provides some additional functionalities, like querying the collected data set. For the samples and alarms API endpoints, both simple and complex query styles are available, whereas for the other endpoints only simple queries are supported.

After validating the query parameters, the processing is done on the database side in the case of most database back ends in order to achieve better performance.

Simple query

Many of the API endpoints accept a query filter argument, which should be a list of data structures that consist of the following items:

- `field`
- `op`
- `value`
- `type`

Regardless of the endpoint on which the filter is applied on, it will always target the fields of the Sample type.

Several fields of the API endpoints accept shorter names than the ones defined in the reference. The API will do the transformation internally and return the output with the fields that are listed in the API reference. The fields are the following:

- `project_id`: project
- `resource_id`: resource
- `user_id`: user

When a filter argument contains multiple constraints of the above form, a logical AND relation between them is implied.

Complex query

The filter expressions of the complex query feature operate on the fields of `Sample`, `Alarm` and `AlarmChange` types. The following comparison operators are supported:

- `=`
- `!=`
- `<`
- `<=`

- >
- >=

The following logical operators can be used:

- `and`
- `or`
- `not`

Note

The `not` operator has different behavior in MongoDB and in the SQLAlchemy-based database engines. If the `not` operator is applied on a non existent metadata field then the result depends on the database engine. In case of MongoDB, it will return every sample as the `not` operator is evaluated true for every sample where the given field does not exist. On the other hand the SQL-based database engine will return an empty result because of the underlying `join` operation.

Complex query supports specifying a list of `orderby` expressions. This means that the result of the query can be ordered based on the field names provided in this list. When multiple keys are defined for the ordering, these will be applied sequentially in the order of the specification. The second expression will be applied on the groups for which the values of the first expression are the same. The ordering can be ascending or descending.

The number of returned items can be bounded using the `limit` option.

The `filter`, `orderby` and `limit` fields are optional.

Note

As opposed to the simple query, complex query is available via a separate API endpoint. For more information see the Telemetry v2 Web API Reference.

Statistics

The sample data can be used in various ways for several purposes, like billing or profiling. In external systems the data is often used in the form of aggregated statistics. The Telemetry API provides several built-in functions to make some basic calculations available without any additional coding.

Telemetry supports the following statistics and aggregation functions:

`avg`	Average of the sample volumes over each period.
`cardinality`	Count of distinct values in each period identified by a key specified as the parameter of this aggregate function. The supported parameter values are: • `project_id` • `resource_id`

- `user_id`

Note

The `aggregate.param` option is required.

`count`	Number of samples in each period.
`max`	Maximum of the sample volumes in each period.
`min`	Minimum of the sample volumes in each period.
`stddev`	Standard deviation of the sample volumes in each period.
`sum`	Sum of the sample volumes over each period.

The simple query and the statistics functionality can be used together in a single API request.

Telemetry command line client and SDK

The Telemetry module provides a command line client, with which the collected data is available just as the alarm definition and retrieval options. The client uses the Telemetry RESTful API in order to execute the requested operations.

To be able to use the **ceilometer** command, the python-ceilometerclient package needs to be installed and configured properly. For details about the installation process, see the Telemetry chapter in the *OpenStack Installation Guide*.

Note

The Telemetry module captures the user-visible resource usage data. Therefore the database will not contain any data without the existence of these resources, like VM images in the OpenStack Image service.

Similarly to other OpenStack command line clients, the **ceilometer** client uses OpenStack Identity for authentication. The proper credentials and *`--auth_url`* parameter have to be defined via command line parameters or environment variables.

This section provides some examples without the aim of completeness. These commands can be used for instance for validating an installation of Telemetry.

To retrieve the list of collected meters, the following command should be used:

```
$ ceilometer meter-list
+------------------------+------------+------+------------------------------------------+----------------------------------
+----------------------------------+
| Name                   | Type       | Unit | Resource ID                              | User ID                          | Project
 ID                               |
+------------------------+------------+------+------------------------------------------+----------------------------------
+----------------------------------+
| cpu                    | cumulative | ns   | bb52e52b-1e42-4751-b3ac-45c52d83ba07     | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| cpu                    | cumulative | ns   | c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b     | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| cpu_util               | gauge      | %    | bb52e52b-1e42-4751-b3ac-45c52d83ba07     | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| cpu_util               | gauge      | %    | c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b     | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.device.read.bytes | cumulative | B    | bb52e52b-1e42-4751-b3ac-45c52d83ba07-hdd | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.device.read.bytes | cumulative | B    | bb52e52b-1e42-4751-b3ac-45c52d83ba07-vda | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.device.read.bytes | cumulative | B    | c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b-hdd | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.device.read.bytes | cumulative | B    | c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b-vda | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| ...
                          |
+------------------------+------------+------+------------------------------------------+----------------------------------
+----------------------------------+
```

The **ceilometer** command was run with `admin` rights, which means that all the data is accessible in the database. For more information about access right see the section called “Users, roles and tenants” [262]. As it can be seen on the above example, there are two VM instances existing in the system, as there are VM instance related meters on the top of the result list. The existence of these meters does not indicate that these instances are running at the time of the request. The result contains the currently collected meters per resource, in an ascending order based on the name of the meter.

Samples are collected for each meter that is present in the list of meters, except in case of instances that are not running or deleted from the OpenStack Compute database. If an instance is no more existing and there is `time_to_live` value is set in the `ceilometer.conf` configuration file, then a group of samples are deleted in each expiration cycle. When the last sample is deleted for a meter, the database can be cleaned up by running `ceilometer-expirer` and the meter will not be present in the list above anymore. For more information about the expiration procedure see the section called “Storing samples” [275].

The Telemetry API supports simple query on the meter endpoint. The query functionality has the following syntax:

```
--query <field1><operator1><value1>;...;<field_n><operator_n><value_n>
```

The following command needs to be invoked to request the meters of one VM instance:

```
$ ceilometer meter-list --query resource=bb52e52b-1e42-4751-b3ac-45c52d83ba07
+-------------------------+------------+----------+--------------------------------------+----------------------------------
+----------------------------------+
| Name                    | Type       | Unit     | Resource ID                          | User ID                          |
 Project ID                         |
+-------------------------+------------+----------+--------------------------------------+----------------------------------
+----------------------------------+
| cpu                     | cumulative | ns       | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| cpu_util                | gauge      | %        | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.ephemeral.size     | gauge      | GB       | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.read.bytes         | cumulative | B        | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.read.bytes.rate    | gauge      | B/s      | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.read.requests      | cumulative | request  | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.read.requests.rate | gauge      | request/s | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.root.size          | gauge      | GB       | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.write.bytes        | cumulative | B        | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.write.bytes.rate   | gauge      | B/s      | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.write.requests     | cumulative | request  | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| disk.write.requests.rate| gauge      | request/s | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| instance                | gauge      | instance | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| instance:m1.tiny        | gauge      | instance | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| memory                  | gauge      | MB       | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
| vcpus                   | gauge      | vcpu     | bb52e52b-1e42-4751-b3ac-45c52d83ba07 | b6e62aad26174382bc3781c12fe413c8 |
 cbfa8e3dfab64a27a87c8e24ecd5c60f |
+-------------------------+------------+----------+--------------------------------------+----------------------------------
+----------------------------------+
```

As it was described above, the whole set of samples can be retrieved that are stored for a meter or filtering the result set by using one of the available query types. The request for all the samples of the `cpu` meter without any additional filtering looks like the following:

```
$ ceilometer sample-list --meter cpu
+--------------------------------------+-------+------------+------------+------+---------------------+
| Resource ID                          | Meter | Type       | Volume     | Unit | Timestamp           |
+--------------------------------------+-------+------------+------------+------+---------------------+
| c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b | cpu   | cumulative | 5.4863e+11 | ns   | 2014-08-31T11:17:03 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu   | cumulative | 5.7848e+11 | ns   | 2014-08-31T11:17:03 |
| c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b | cpu   | cumulative | 5.4811e+11 | ns   | 2014-08-31T11:07:05 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu   | cumulative | 5.7797e+11 | ns   | 2014-08-31T11:07:05 |
| c8d2e153-a48f-4cec-9e93-86e7ac6d4b0b | cpu   | cumulative | 5.3589e+11 | ns   | 2014-08-31T10:27:19 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu   | cumulative | 5.6397e+11 | ns   | 2014-08-31T10:27:19 |
| ...                                                                                                 |
+--------------------------------------+-------+------------+------------+------+---------------------+
```

The result set of the request contains the samples for both instances ordered by the timestamp field in the default descending order.

The simple query makes it possible to retrieve only a subset of the collected samples. The following command can be executed to request the `cpu` samples of only one of the VM instances:

```
$ ceilometer sample-list --meter cpu --query resource=bb52e52b-1e42-4751-b3ac-45c52d83ba07
+--------------------------------------+------+------------+------------+------+---------------------+
| Resource ID                          | Name | Type       | Volume     | Unit | Timestamp           |
+--------------------------------------+------+------------+------------+------+---------------------+
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu  | cumulative | 5.7906e+11 | ns   | 2014-08-31T11:27:08 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu  | cumulative | 5.7848e+11 | ns   | 2014-08-31T11:17:03 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu  | cumulative | 5.7797e+11 | ns   | 2014-08-31T11:07:05 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu  | cumulative | 5.6397e+11 | ns   | 2014-08-31T10:27:19 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu  | cumulative | 5.6207e+11 | ns   | 2014-08-31T10:17:03 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu  | cumulative | 5.3831e+11 | ns   | 2014-08-31T08:41:57 |
| ...                                                                                                |
+--------------------------------------+------+------------+------------+------+---------------------+
```

As it can be seen on the output above, the result set contains samples for only one instance of the two.

The **ceilometer query-samples** command is used to execute rich queries. This command accepts the following parameters:

--filter	Contains the filter expression for the query in the form of: `{complex_op: [{simple_op: {field_name: value}}]}`.
--orderby	Contains the list of `orderby` expressions in the form of: `[{field_name: direction}, {field_name: direction}]`.
--limit	Specifies the maximum number of samples to return.

For more information about complex queries see the section called “Complex query” [277].

As the complex query functionality provides the possibility of using complex operators, it is possible to retrieve a subset of samples for a given VM instance. To request for the first six samples for the `cpu` and `disk.read.bytes` meters, the following command should be invoked:

```
$ ceilometer query-samples --filter '{"and": \
  [{"=":{"resource":"bb52e52b-1e42-4751-b3ac-45c52d83ba07"}},{"or":[{"=":{"counter_name":"cpu"}}, \
  {"=":{"counter_name":"disk.read.bytes"}}]}]}' --orderby '[{"timestamp":"asc"}]' --limit 6
+--------------------------------------+-----------------+------------+------------+------+---------------------+
| Resource ID                          | Meter           | Type       | Volume     | Unit | Timestamp           |
+--------------------------------------+-----------------+------------+------------+------+---------------------+
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | disk.read.bytes | cumulative | 385334.0   | B    | 2014-08-30T13:00:46 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu             | cumulative | 1.2132e+11 | ns   | 2014-08-30T13:00:47 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu             | cumulative | 1.4295e+11 | ns   | 2014-08-30T13:10:51 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | disk.read.bytes | cumulative | 601438.0   | B    | 2014-08-30T13:10:51 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | disk.read.bytes | cumulative | 601438.0   | B    | 2014-08-30T13:20:33 |
| bb52e52b-1e42-4751-b3ac-45c52d83ba07 | cpu             | cumulative | 1.4795e+11 | ns   | 2014-08-30T13:20:34 |
+--------------------------------------+-----------------+------------+------------+------+---------------------+
```

Telemetry python bindings

The command line client library provides python bindings in order to use the Telemetry Python API directly from python programs.

The first step in setting up the client is to create a client instance with the proper credentials:

```
>>> import ceilometerclient.client
>>> cclient = ceilometerclient.client.get_client(VERSION, username=USERNAME,
 password=PASSWORD, tenant_name=PROJECT_NAME, auth_url=AUTH_URL)
```

The `VERSION` parameter can be `1` or `2`, specifying the API version to be used.

The method calls look like the following:

```
>>> cclient.meters.list()
 [<Meter ...>, ...]

>>> cclient.samples.list()
 [<Sample ...>, ...]
```

For further details about the python-ceilometerclient package, see the Python bindings to the OpenStack Ceilometer API reference.

Publishers

The Telemetry module provides several transport methods to forward the data collected to the `ceilometer-collector` service or to an external system. The consumers of this data are widely different, like monitoring systems, for which data loss is acceptable and billing systems, which require reliable data transportation. Telemetry provides methods to fulfill the requirements of both kind of systems, as it is described below.

The publisher component makes it possible to persist the data into storage through the message bus or to send it to one or more external consumers. One chain can contain multiple publishers.

To solve the above mentioned problem, the notion of multi-publisher can be configured for each datapoint within the Telemetry module, allowing the same technical meter or event to be published multiple times to multiple destinations, each potentially using a different transport.

Publishers can be specified in the `publishers` section for each pipeline (for further details about pipelines see the section called “Data collection and processing” [269]) that is defined in the file called `pipeline.yaml`.

The following publisher types are supported:

notifier	It can be specified in the form of `notifier://?option1=value1&option2=value2`. It emits data over AMQP using oslo.messaging. This is the recommended method of publishing.
rpc	It can be specified in the form of `rpc://?option1=value1&option2=value2`. It emits metering data over lossy AMQP. This method is synchronous and may experience performance issues.
udp	It can be specified in the form of `udp://<host>:<port>/`. It emits metering data for over UDP.
file	It can be specified in the form of `file://path?option1=value1&option2=value2`. This publisher records metering data into a file.

Note

If a file name and location is not specified, this publisher does not log any meters, instead it logs a warning message in the configured log file for Telemetry.

kafka It can be specified in the form of `kafka://`*`kafka_broker_ip`*`:`*`kafka_broker_port`*`?topic=kafka_topic &option1=value1`. This publisher sends metering data to a kafka broker.

Note

If the topic parameter is missing, this publisher brings out metering data under a topic name, `ceilometer`. When the port number is not specified, this publisher uses 9092 as the broker's port.

The following options are available for `rpc` and `notifier`. The policy option can be used by `kafka` publisher:

`per_meter_topic` The value of it is 1. It is used for publishing the samples on additional `metering_topic.sample_name` topic queue besides the default `metering_topic` queue.

`policy` It is used for configuring the behavior for the case, when the publisher fails to send the samples, where the possible predefined values are the following:

default	Used for waiting and blocking until the samples have been sent.
drop	Used for dropping the samples which are failed to be sent.
queue	Used for creating an in-memory queue and retrying to send the samples on the queue on the next samples publishing period (the queue length can be configured with `max_queue_length`, where 1024 is the default value).

The following options are available for the `file` publisher:

`max_bytes` When this option is greater than zero, it will cause a rollover. When the size is about to be exceeded, the file is closed and a new file is silently opened for output. If its value is zero, rollover never occurs.

`backup_count` If this value is non-zero, an extension will be appended to the filename of the old log, as '.1', '.2', and so forth until the specified value is reached. The file that is written and contains the newest data is always the one that is specified without any extensions.

The default publisher is `notifier`, without any additional options specified. A sample `publishers` section in the `/etc/ceilometer/pipeline.yaml` looks like the following:

```
publishers:
    - udp://10.0.0.2:1234
    - rpc://?per_meter_topic=1
    - notifier://?policy=drop&max_queue_length=512
```

Alarms

Alarms provide user-oriented Monitoring-as-a-Service for resources running on OpenStack. This type of monitoring ensures you can automatically scale in or out a group of instances through the Orchestration module, but you can also use alarms for general-purpose awareness of your cloud resources' health.

These alarms follow a tri-state model:

ok	The rule governing the alarm has been evaluated as `False`.
alarm	The rule governing the alarm have been evaluated as `True`.
insufficient data	There are not enough datapoints available in the evaluation periods to meaningfully determine the alarm state.

Alarm definitions

The definition of an alarm provides the rules that govern when a state transition should occur, and the actions to be taken thereon. The nature of these rules depend on the alarm type.

Threshold rule alarms

For conventional threshold-oriented alarms, state transitions are governed by:

- A static threshold value with a comparison operator such as greater than or less than.
- A statistic selection to aggregate the data.
- A sliding time window to indicate how far back into the recent past you want to look.

Combination rule alarms

The Telemetry module also supports the concept of a meta-alarm, which aggregates over the current state of a set of underlying basic alarms combined via a logical operator (AND or OR).

Alarm dimensioning

A key associated concept is the notion of *dimensioning* which defines the set of matching meters that feed into an alarm evaluation. Recall that meters are per-resource-instance, so in the simplest case an alarm might be defined over a particular meter applied to all resources visible to a particular user. More useful however would be the option to explicitly select which specific resources you are interested in alarming on.

At one extreme you might have narrowly dimensioned alarms where this selection would have only a single target (identified by resource ID). At the other extreme, you could have widely dimensioned alarms where this selection identifies many resources over which the statistic is aggregated. For example all instances booted from a particular image or all instances with matching user metadata (the latter is how the Orchestration module identifies autoscaling groups).

Alarm evaluation

Alarms are evaluated by the `alarm-evaluator` service on a periodic basis, defaulting to once every minute.

Alarm actions

Any state transition of individual alarm (to `ok`, `alarm`, or `insufficient data`) may have one or more actions associated with it. These actions effectively send a signal to a consumer that the state transition has occurred, and provide some additional context. This includes the new and previous states, with some reason data describing the disposition with respect to the threshold, the number of datapoints involved and most recent of these. State transitions are detected by the `alarm-evaluator`, whereas the `alarm-notifier` effects the actual notification action.

Webhooks

These are the *de facto* notification type used by Telemetry alarming and simply involve a HTTP POST request being sent to an endpoint, with a request body containing a description of the state transition encoded as a JSON fragment.

Log actions

These are a lightweight alternative to webhooks, whereby the state transition is simply logged by the `alarm-notifier`, and are intended primarily for testing purposes.

Workload partitioning

The alarm evaluation process uses the same mechanism for workload partitioning as the central and compute agents. The Tooz library provides the coordination within the groups of service instances. For further information about this approach see the section called “Support for HA deployment of the central and compute agent services” [267].

To use this workload partitioning solution set the `evaluation_service` option to `default`. For more information, see the alarm section in the *OpenStack Configuration Reference*.

Using alarms

The **ceilometer** CLI provides simple verbs for creating and manipulating alarms.

Alarm creation

An example of creating a threshold-oriented alarm, based on an upper bound on the CPU utilization for a particular instance:

```
$ ceilometer alarm-threshold-create --name cpu_hi \
  --description 'instance running hot' \
  --meter-name cpu_util --threshold 70.0 \
  --comparison-operator gt --statistic avg \
  --period 600 --evaluation-periods 3 \
  --alarm-action 'log://' \
  --query resource_id=INSTANCE_ID
```

This creates an alarm that will fire when the average CPU utilization for an individual instance exceeds 70% for three consecutive 10 minute periods. The notification in this case is simply a log message, though it could alternatively be a webhook URL.

Note

Alarm names must be unique for the alarms associated with an individual project.

The cloud administrator can limit the maximum resulting actions for three different states, and the ability for a normal user to create `log://` and `test://` notifiers is disabled. This prevents unintentional consumption of disk and memory resources by the Telemetry service.

The sliding time window over which the alarm is evaluated is 30 minutes in this example. This window is not clamped to wall-clock time boundaries, rather it's anchored on the current time for each evaluation cycle, and continually creeps forward as each evaluation cycle rolls around (by default, this occurs every minute).

The period length is set to 600s in this case to reflect the out-of-the-box default cadence for collection of the associated meter. This period matching illustrates an important general principal to keep in mind for alarms:

Note

The alarm period should be a whole number multiple (1 or more) of the interval configured in the pipeline corresponding to the target meter.

Otherwise the alarm will tend to flit in and out of the `insufficient data` state due to the mismatch between the actual frequency of datapoints in the metering store and the statistics queries used to compare against the alarm threshold. If a shorter alarm period is needed, then the corresponding interval should be adjusted in the `pipeline.yaml` file.

Other notable alarm attributes that may be set on creation, or via a subsequent update, include:

state	The initial alarm state (defaults to `insufficient data`).
description	A free-text description of the alarm (defaults to a synopsis of the alarm rule).
enabled	True if evaluation and actioning is to be enabled for this alarm (defaults to True).
repeat-actions	True if actions should be repeatedly notified while the alarm remains in the target state (defaults to False).
ok-action	An action to invoke when the alarm state transitions to `ok`.
insufficient-data-action	An action to invoke when the alarm state transitions to `insufficient data`.

time-constraint	Used to restrict evaluation of the alarm to certain times of the day or days of the week (expressed as `cron` expression with an optional timezone).

An example of creating a combination alarm, based on the combined state of two underlying alarms:

```
$ ceilometer alarm-combination-create --name meta \
  --alarm_ids ALARM_ID1 \
  --alarm_ids ALARM_ID2 \
  --operator or \
  --alarm-action 'http://example.org/notify'
```

This creates an alarm that will fire when ether one of two underlying alarms transition into the alarm state. The notification in this case is a webhook call. Any number of underlying alarms can be combined in this way, using either `and` or `or`.

Alarm retrieval

You can display all your alarms via (some attributes are omitted for brevity):

```
$ ceilometer alarm-list
+----------+--------+-------------------+---------------------------------+
| Alarm ID | Name   | State             | Alarm condition                 |
+----------+--------+-------------------+---------------------------------+
| ALARM_ID | cpu_hi | insufficient data | cpu_util > 70.0 during 3 x 600s |
+----------+--------+-------------------+---------------------------------+
```

In this case, the state is reported as `insufficient data` which could indicate that:

- meters have not yet been gathered about this instance over the evaluation window into the recent past (for example a brand-new instance)
- *or*, that the identified instance is not visible to the user/tenant owning the alarm
- *or*, simply that an alarm evaluation cycle hasn't kicked off since the alarm was created (by default, alarms are evaluated once per minute).

Note

The visibility of alarms depends on the role and project associated with the user issuing the query:

- admin users see *all* alarms, regardless of the owner
- non-admin users see only the alarms associated with their project (as per the normal tenant segregation in OpenStack)

Alarm update

Once the state of the alarm has settled down, we might decide that we set that bar too low with 70%, in which case the threshold (or most any other alarm attribute) can be updated thusly:

```
$ ceilometer alarm-update --threshold 75 ALARM_ID
```

The change will take effect from the next evaluation cycle, which by default occurs every minute.

Most alarm attributes can be changed in this way, but there is also a convenient short-cut for getting and setting the alarm state:

```
$ ceilometer alarm-state-get ALARM_ID
$ ceilometer alarm-state-set --state ok -a ALARM_ID
```

Over time the state of the alarm may change often, especially if the threshold is chosen to be close to the trending value of the statistic. You can follow the history of an alarm over its lifecycle via the audit API:

```
$ ceilometer alarm-history ALARM_ID
+------------------+-----------+---------------------------------------+
| Type             | Timestamp | Detail                                |
+------------------+-----------+---------------------------------------+
| creation         | time0     | name: cpu_hi                          |
|                  |           | description: instance running hot     |
|                  |           | type: threshold                       |
|                  |           | rule: cpu_util > 70.0 during 3 x 600s |
| state transition | time1     | state: ok                             |
| rule change      | time2     | rule: cpu_util > 75.0 during 3 x 600s |
+------------------+-----------+---------------------------------------+
```

Alarm deletion

An alarm that's no longer required can be disabled so that it is no longer actively evaluated:

```
$ ceilometer alarm-update --enabled False -a ALARM_ID
```

or even deleted permanently (an irreversible step):

```
$ ceilometer alarm-delete ALARM_ID
```

Note

By default, alarm history is retained for deleted alarms.

Measurements

The Telemetry module collects meters within an OpenStack deployment. This section provides a brief summary about meters format and origin and also contains the list of available meters.

Telemetry collects meters by polling the infrastructure elements and also by consuming the notifications emitted by other OpenStack services. For more information about the polling mechanism and notifications see the section called "Data collection" [262]. There are several meters which are collected by polling and by consuming. The origin for each meter is listed in the tables below.

Note

You may need to configure Telemetry or other OpenStack services in order to be able to collect all the samples you need. For further information about configuration requirements see the Telemetry chapter in the *OpenStack Installation Guide*. Also check the Telemetry manual installation description.

Telemetry uses the following meter types:

Table 8.4. Telemetry meter types

Type	Description
Cumulative	Increasing over time (instance hours)
Delta	Changing over time (bandwidth)
Gauge	Discrete items (floating IPs, image uploads) and fluctuating values (disk I/O)

Telemetry provides the possibility to store metadata for samples. This metadata can be extended for OpenStack Compute and OpenStack Object Storage.

In order to add additional metadata information to OpenStack Compute you have two options to choose from. The first one is to specify them when you boot up a new instance. The additional information will be stored with the sample in the form of `resource_metadata.user_metadata.*`. The new field should be defined by using the prefix `metering.`. The modified boot command look like the following:

```
$ nova boot --meta metering.custom_metadata=a_value my_vm
```

The other option is to set the `reserved_metadata_keys` to the list of metadata keys that you would like to be included in `resource_metadata` of the instance related samples that are collected for OpenStack Compute. This option is included in the `DEFAULT` section of the `ceilometer.conf` configuration file.

You might also specify headers whose values will be stored along with the sample data of OpenStack Object Storage. The additional information is also stored under `resource_metadata`. The format of the new field is `resource_metadata.http_header_$name`, where `$name` is the name of the header with `-` replaced by `_`.

For specifying the new header, you need to set `metadata_headers` option under the `[filter:ceilometer]` section in `proxy-server.conf` under the `swift` folder. You can use this additional data for instance to distinguish external and internal users.

The list of measurements is grouped by services which are polled by Telemetry or emits notifications that this module consumes.

Note

The Telemetry module supports storing notifications as events. This functionality was added later, therefore the list of meters still contains existence type and other event related items. The proper way of using Telemetry is to configure it to use the event store and turn off the collection of the event related meters. For further information about events see Events section in the Telemetry documentation. For further information about how to turn on and off meters see the section called "Pipeline configuration" [270]. Please also note that currently no migration is available to move the already existing event type samples to the event store.

OpenStack Compute

The following meters are collected for OpenStack Compute:

Table 8.5. OpenStack Compute meters

Name	Type	Unit	Resource	Origin	Support	Note
Meters added in the Icehouse release or earlier						
instance	Gauge	instance	instance ID	Notification, Pollster	Libvirt, Hyper-V, vSphere	Existence of instance.
instance:<type>	Gauge	instance	instance ID	Notification, Pollster	Libvirt, Hyper-V, vSphere	Existence of instance <type> (OpenStack types).
memory	Gauge	MB	instance ID	Notification	Libvirt, Hyper-V	Volume of RAM allocated to the instance.
memory.usage	Gauge	MB	instance ID	Pollster	vSphere	Volume of RAM used by the instance from the amount of its allocated memory.
cpu	Cumulative	ns	instance ID	Pollster	Libvirt, Hyper-V	CPU time used.
cpu_util	Gauge	%	instance ID	Pollster	Libvirt, Hyper-V, vSphere	Average CPU utilisation.
vcpus	Gauge	vcpu	instance ID	Notification	Libvirt, Hyper-V	Number of virtual CPUs allocated to the instance.
disk.read.requests	Cumulative	request	instance ID	Pollster	Libvirt, Hyper-V	Number of read requests.
disk.read.requests.rate	Gauge	request/s	instance ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of read requests.
disk.write.requests	Cumulative	request	instance ID	Pollster	Libvirt, Hyper-V	Number of write requests.
disk.write.requests.rate	Gauge	request/s	instance ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of write requests.
disk.read.bytes	Cumulative	B	instance ID	Pollster	Libvirt, Hyper-V	Volume of reads.
disk.read.bytes.rate	Gauge	B/s	instance ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of reads.
disk.write.bytes	Cumulative	B	instance ID	Pollster	Libvirt, Hyper-V	Volume of writes.
disk.write.bytes.rate	Gauge	B/s	instance ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of writes.
disk.root.size	Gauge	GB	instance ID	Notification	Libvirt, Hyper-V	Size of root disk.
disk.ephemeral.size	Gauge	GB	instance ID	Notification	Libvirt, Hyper-V	Size of ephemeral disk.
network.incoming.bytes	Cumulative	B	interface ID	Pollster	Libvirt, Hyper-V	Number of incoming bytes.

Name	Type	Unit	Resource	Origin	Support	Note
network.incoming.bytes.rate	Gauge	B/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of incoming bytes.
network.outgoing.bytes	Cumulative	B	interface ID	Pollster	Libvirt, Hyper-V	Number of outgoing bytes.
network.outgoing.bytes.rate	Gauge	B/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of outgoing bytes.
network.incoming.packets	Cumulative	packet	interface ID	Pollster	Libvirt, Hyper-V	Number of incoming packets.
network.incoming.packets.rate	Gauge	packet/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of incoming packets.
network.outpoing.packets	Cumulative	packet	interface ID	Pollster	Libvirt, Hyper-V	Number of outgoing packets.
network.outgoing.packets.rate	Gauge	packet/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of outgoing packets.
Meters added or hypervisor support changed in the Juno release						
instance	Gauge	instance	instance ID	Notification, Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Existence of instance.
instance:<type>	Gauge	instance	instance ID	Notification, Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Existence of instance <type> (OpenStack types).
memory.usage	Gauge	MB	instance ID	Pollster	vSphere, XenAPI	Volume of RAM used by the instance from the amount of its allocated memory.
cpu_util	Gauge	%	instance ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average CPU utilisation.
disk.read.bytes.rate	Gauge	B/s	instance ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average rate of reads.
disk.write.bytes.rate	Gauge	B/s	instance ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average rate of writes.
disk.device.read.requests	Cumulative	request	disk ID	Pollster	Libvirt, Hyper-V	Number of read requests.
disk.device.read.requests.rate	Gauge	request/s	disk ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of read requests.

Name	Type	Unit	Resource	Origin	Support	Note
disk.device.write.requests	Cumulative	request	disk ID	Pollster	Libvirt, Hyper-V	Number of write requests.
disk.device.write.requests.rate	Gauge	request/s	disk ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of write requests.
disk.device.read.bytes	Cumulative	B	disk ID	Pollster	Libvirt, Hyper-V	Volume of reads.
disk.device.read.bytes.rate	Gauge	B/s	disk ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of reads.
disk.device.write.bytes	Cumulative	B	disk ID	Pollster	Libvirt, Hyper-V	Volume of writes.
disk.device.write.bytes.rate	Gauge	B/s	disk ID	Pollster	Libvirt, Hyper-V, vSphere	Average rate of writes.
network.incoming.bytes.rate	Gauge	B/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average rate of incoming bytes.
network.outgoing.bytes.rate	Gauge	B/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average rate of outgoing bytes.
network.incoming.packets.rate	Gauge	packet/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average rate of incoming packets.
network.outgoing.packets.rate	Gauge	packet/s	interface ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Average rate of outgoing packets.
Meters added or hypervisor support changed in the Kilo release						
memory.usage	Gauge	MB	instance ID	Pollster	Libvirt, Hyper-V, vSphere, XenAPI	Volume of RAM used by the instance from the amount of its allocated memory.
memory.resident	Gauge	MB	instance ID	Pollster	Libvirt	Volume of RAM used by the instance on the physical machine.
disk.latency	Gauge	ms	instance ID	Pollster	Hyper-V	Average disk latency.
disk.iops	Gauge	count/s	instance ID	Pollster	Hyper-V	Average disk iops.
disk.device.latency	Gauge	ms	disk ID	Pollster	Hyper-V	Average disk latency per device.
disk.device.iops	Gauge	count/s	disk ID	Pollster	Hyper-V	Average disk iops per device.
disk.capacity	Gauge	B	instance ID	Pollster	Libvirt	The amount of disk that the instance can see.

Name	Type	Unit	Resource	Origin	Support	Note
disk.allocation	Gauge	B	instance ID	Pollster	Libvirt	The amount of disk occupied by the instance on the host machine.
disk.usage	Gauge	B	instance ID	Pollster	Libvirt	The physical size in bytes of the image container on the host.
disk.device.capacity	Gauge	B	disk ID	Pollster	Libvirt	The amount of disk per device that the instance can see.
disk.device.allocation	Gauge	B	disk ID	Pollster	Libvirt	The amount of disk per device occupied by the instance on the host machine.
disk.device.usage	Gauge	B	disk ID	Pollster	Libvirt	The physical size in bytes of the image container on the host per device.

The Telemetry module supports to create new meters by using transformers. For more details about transformers see the section called "Transformers" [272]. Among the meters gathered from libvirt and Hyper-V there are a few ones which are generated from other meters. The list of meters that are created by using the `rate_of_change` transformer from the above table is the following:

- cpu_util
- disk.read.requests.rate
- disk.write.requests.rate
- disk.read.bytes.rate
- disk.write.bytes.rate
- disk.device.read.requests.rate
- disk.device.write.requests.rate
- disk.device.read.bytes.rate
- disk.device.write.bytes.rate
- network.incoming.bytes.rate
- network.outgoing.bytes.rate
- network.incoming.packets.rate
- network.outgoing.packets.rate

Note

To enable the libvirt `memory.usage` support, you need to install libvirt version 1.1.1+, QEMU version 1.5+, and you also need to prepare suitable balloon driver in the image. It is applicable particularly for Windows guests, most modern Linux distributions already have it built in. Telemetry is not able to fetch the `memory.usage` samples without the image balloon driver.

OpenStack Compute is capable of collecting CPU related meters from the compute host machines. In order to use that you need to set the `compute_monitors` option to `ComputeDriverCPUMonitor` in the `nova.conf` configuration file. For further information see the Compute configuration section in the Compute chapter of the *OpenStack Configuration Reference*.

The following host machine related meters are collected for OpenStack Compute:

Table 8.6. OpenStack Compute host meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
compute.node.cpu.frequency	Gauge	MHz	host ID	Notification	CPU frequency.
compute.node.cpu.kernel.time	Cumulative	ns	host ID	Notification	CPU kernel time.
compute.node.cpu.idle.time	Cumulative	ns	host ID	Notification	CPU idle time.
compute.node.cpu.user.time	Cumulative	ns	host ID	Notification	CPU user mode time.
compute.node.cpu.iowait.time	Cumulative	ns	host ID	Notification	CPU I/O wait time.
compute.node.cpu.kernel.percent	Gauge	%	host ID	Notification	CPU kernel percentage.
compute.node.cpu.idle.percent	Gauge	%	host ID	Notification	CPU idle percentage.
compute.node.cpu.user.percent	Gauge	%	host ID	Notification	CPU user mode percentage.
compute.node.cpu.iowait.percent	Gauge	%	host ID	Notification	CPU I/O wait percentage.
compute.node.cpu.percent	Gauge	%	host ID	Notification	CPU utilisation.

Bare metal service

Telemetry captures notifications that are emitted by the Bare metal service. The source of the notifications are IPMI sensors that collect data from the host machine.

Note

The sensor data is not available in the Bare metal service by default. To enable the meters and configure this module to emit notifications about the measured values see the Installation Guide for the Bare metal service.

The following meters are recorded for the Bare metal service:

Table 8.7. Metrics of Bare metal module for OpenStack

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					
hardware.ipmi.fan	Gauge	RPM	fan sensor	Notification	Fan rounds per minute (RPM).
hardware.ipmi.temperature	Gauge	C	temperature sensor	Notification	Temperate reading from sensor.
hardware.ipmi.current	Gauge	W	current sensor	Notification	Current reading from sensor.
hardware.ipmi.voltage	Gauge	V	voltage sensor	Notification	Voltage reading from sensor.

IPMI based meters

Another way of gathering IPMI based data is to use IPMI sensors independently from the Bare metal service's components. Same meters as the section called “Bare metal service” [294] could be fetched except that origin is `Pollster` instead of `Notification`.

You need to deploy the `ceilometer-agent-ipmi` on each IPMI-capable node in order to poll local sensor data. For further information about the IPMI agent see the section called “IPMI agent” [268].

Warning

To avoid duplication of metering data and unnecessary load on the IPMI interface, do not deploy the IPMI agent on nodes that are managed by the Bare metal service and keep the `conductor.send_sensor_data` option set to `False` in the `ironic.conf` configuration file.

Besides generic IPMI sensor data, the following Intel Node Manager meters are recorded from capable platform:

Table 8.8. IPMI based meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					
hardware.ipmi.node.power	Gauge	W	host ID	Pollster	Current power of the system.
hardware.ipmi.node.temperature	Gauge	C	host ID	Pollster	Current temperature of the system.
Meters added in the Kilo release					
hardware.ipmi.node.inlet_temperature	Gauge	C	host ID	Pollster	Inlet temperature of the system.
hardware.ipmi.node.outlet_temperature	Gauge	C	host ID	Pollster	Outlet temperature of the system.
hardware.ipmi.node.airflow	Gauge	CFM	host ID	Pollster	Volumetric airflow of the system, expressed as 1/10th of CFM.
hardware.ipmi.node.cups	Gauge	CUPS	host ID	Pollster	CUPS(Compute Usage Per Second) index data of the system.
hardware.ipmi.node.cpu_util	Gauge	%	host ID	Pollster	CPU CUPS utilization of the system.
hardware.ipmi.node.mem_util	Gauge	%	host ID	Pollster	Memory CUPS utilization of the system.
hardware.ipmi.node.io_util	Gauge	%	host ID	Pollster	IO CUPS utilization of the system.
Meters renamed in the Kilo release					
Original Name			New Name		
hardware.ipmi.node.temperature			hardware.ipmi.node.inlet_temperature		

SNMP based meters

Telemetry supports gathering SNMP based generic host meters. In order to be able to collect this data you need to run `smpd` on each target host.

The following meters are available about the host machines by using SNMP:

Table 8.9. SNMP based meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Kilo release					
hardware.cpu.load.1min	Gauge	process	host ID	Pollster	CPU load in the past 1 minute.
hardware.cpu.load.5min	Gauge	process	host ID	Pollster	CPU load in the past 5 minutes.
hardware.cpu.load.10min	Gauge	process	host ID	Pollster	CPU load in the past 10 minutes.
hardware.disk.size.total	Gauge	B	disk ID	Pollster	Total disk size.
hardware.disk.size.used	Gauge	B	disk ID	Pollster	Used disk size.
hardware.memory.total	Gauge	B	host ID	Pollster	Total physical memory size.
hardware.memory.used	Gauge	B	host ID	Pollster	Used physical memory size.
hardware.memory.buffer	Gauge	B	host ID	Pollster	Physical memory buffer size.
hardware.memory.cached	Gauge	B	host ID	Pollster	Cached physical memory size.
hardware.memory.swap.total	Gauge	B	host ID	Pollster	Total swap space size.
hardware.memory.swap.avail	Gauge	B	host ID	Pollster	Available swap space size.
hardware.network.incoming.bytes	Cumulative	B	interface ID	Pollster	Bytes received by network interface.
hardware.network.outgoing.bytes	Cumulative	B	interface ID	Pollster	Bytes sent by network interface.
hardware.network.outgoing.errors	Cumulative	packet	interface ID	Pollster	Sending error of network interface.
hardware.network.ip.incoming.datagrams	Cumulative	datagrams	host ID	Pollster	Number of received datagrams.
hardware.network.ip.outgoing.datagrams	Cumulative	datagrams	host ID	Pollster	Number of sent datagrams.
hardware.system_stats.io.incoming.blocks	Cumulative	blocks	host ID	Pollster	Aggregated number of blocks received to block device.
hardware.system_stats.io.outgoing.blocks	Cumulative	blocks	host ID	Pollster	Aggregated number of blocks sent to block device.
hardware.system_stats.cpu.idle	Gauge	%	host ID	Pollster	CPU idle percentage.

OpenStack Image service

The following meters are collected for OpenStack Image service:

Table 8.10. OpenStack Image Service meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
image	Gauge	image	image ID	Notification, Pollster	Existence of the image.

Name	Type	Unit	Resource	Origin	Note
image.size	Gauge	image	image ID	Notification, Pollster	Size of the uploaded image.
image.update	Delta	image	image ID	Notification	Number of updates on the image.
image.upload	Delta	image	image ID	Notification	Number of uploads on the image.
image.delete	Delta	image	image ID	Notification	Number of deletes on the image.
image.download	Delta	B	image ID	Notification	Image is downloaded.
image.serve	Delta	B	image ID	Notification	Image is served out.

OpenStack Block Storage

The following meters are collected for OpenStack Block Storage:

Table 8.11. OpenStack Block Storage meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
volume	Gauge	volume	volume ID	Notification	Existence of the volume.
volume.size	Gauge	GB	volume ID	Notification	Size of the volume.
Meters added in the Juno release					
snapshot	Gauge	snapshot	snapshot ID	Notification	Existence of the snapshot.
snapshot.size	Gauge	GB	snapshot ID	Notification	Size of the snapshot.
Meters added in the Kilo release					
volume.create.(start|end)	Delta	volume	volume ID	Notification	Creation of the volume.
volume.delete.(start|end)	Delta	volume	volume ID	Notification	Deletion of the volume.
volume.update.(start|end)	Delta	volume	volume ID	Notification	Update the name or description of the volume.
volume.resize.(start|end)	Delta	volume	volume ID	Notification	Update the size of the volume.
volume.attach.(start|end)	Delta	volume	volume ID	Notification	Attaching the volume to an instance.
volume.detach.(start|end)	Delta	volume	volume ID	Notification	Detaching the volume from an instance.
snapshot.create.(start|end)	Delta	snapshot	snapshot ID	Notification	Creation of the snapshot.
snapshot.delete.(start|end)	Delta	snapshot	snapshot ID	Notification	Deletion of the snapshot.

OpenStack Object Storage

The following meters are collected for OpenStack Object Storage:

Table 8.12. OpenStack Object Storage meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
storage.objects	Gauge	object	storage ID	Pollster	Number of objects.
storage.objects.size	Gauge	B	storage ID	Pollster	Total size of stored objects.
storage.objects.containers	Gauge	container	storage ID	Pollster	Number of containers.
storage.objects.incoming.bytes	Delta	B	storage ID	Notification	Number of incoming bytes.
storage.objects.outgoing.bytes	Delta	B	storage ID	Notification	Number of outgoing bytes.

Name	Type	Unit	Resource	Origin	Note
storage.api.request	Delta	request	storage ID	Notification	Number of API requests against OpenStack Object Storage.
storage.containers.objects	Gauge	object	storage ID/ container	Pollster	Number of objects in container.
storage.containers.objects.size	Gauge	B	storage ID/ container	Pollster	Total size of stored objects in container.

Ceph Object Storage

In order to gather meters from Ceph, you have to install and configure the Ceph Object Gateway (radosgw) as it is described in the Installation Manual. You have to enable usage logging in order to get the related meters from Ceph. You will also need an `admin` user with `users`, `buckets`, `metadata` and `usage` `caps` configured.

In order to access Ceph from Telemetry, you need to specify a `service group` for `radosgw` in the `ceilometer.conf` configuration file along with `access_key` and `secret_key` of the `admin` user mentioned above.

The following meters are collected for Ceph Object Storage:

Table 8.13. Metrics for Ceph Object Storage

Name	Type	Unit	Resource	Origin	Note
Meters added in the Kilo release					
radosgw.objects	Gauge	object	storage ID	Pollster	Number of objects.
radosgw.objects.size	Gauge	B	storage ID	Pollster	Total size of stored objects.
radosgw.objects.containers	Gauge	container	storage ID	Pollster	Number of containers.
radosgw.api.request	Gauge	request	storage ID	Pollster	Number of API requests against Ceph Object Gateway (radosgw).
radosgw.containers.objects	Gauge	object	storage ID/ container	Pollster	Number of objects in container.
radosgw.containers.objects.size	Gauge	B	storage ID/ container	Pollster	Total size of stored objects in container.

Note

The `usage` related information may not be updated right after an upload or download, because the Ceph Object Gateway needs time to update the usage properties. For instance, the default configuration needs approximately 30 minutes to generate the usage logs.

OpenStack Identity

The following meters are collected for OpenStack Identity:

Table 8.14. OpenStack Identity meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					
identity.authenticate.success	Delta	user	user ID	Notification	User successfully authenticated.

Name	Type	Unit	Resource	Origin	Note
identity.authenticate.pending	Delta	user	user ID	Notification	User pending authentication.
identity.authenticate.failure	Delta	user	user ID	Notification	User failed to authenticate.
identity.user.created	Delta	user	user ID	Notification	User is created.
identity.user.deleted	Delta	user	user ID	Notification	User is deleted.
identity.user.updated	Delta	user	user ID	Notification	User is updated.
identity.group.created	Delta	group	group ID	Notification	Group is created.
identity.group.deleted	Delta	group	group ID	Notification	Group is deleted.
identity.group.updated	Delta	group	group ID	Notification	Group is updated.
identity.role.created	Delta	role	role ID	Notification	Role is created.
identity.role.deleted	Delta	role	role ID	Notification	Role is deleted.
identity.role.updated	Delta	role	role ID	Notification	Role is updated.
identity.project.created	Delta	project	project ID	Notification	Project is created.
identity.project.deleted	Delta	project	project ID	Notification	Project is deleted.
identity.project.updated	Delta	project	project ID	Notification	Project is updated.
identity.trust.created	Delta	trust	trust ID	Notification	Trust is created.
identity.trust.deleted	Delta	trust	trust ID	Notification	Trust is deleted.
Meters added in the Kilo release					
identity.role_assignment.created	Delta	role_assignment	role ID	Notification	Role is added to an actor on a target.
identity.role_assignment.deleted	Delta	role_assignment	role ID	Notification	Role is removed from an actor on a target.

OpenStack Networking

The following meters are collected for OpenStack Networking:

Table 8.15. OpenStack Networking meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
network	Gauge	network	network ID	Notification	Existence of network.
network.create	Delta	network	network ID	Notification	Creation requests for this network.
network.update	Delta	network	network ID	Notification	Update requests for this network.
subnet	Gauge	subnet	subnet ID	Notification	Existence of subnet.
subnet.create	Delta	subnet	subnet ID	Notification	Creation requests for this subnet.
subnet.update	Delta	subnet	subnet ID	Notification	Update requests for this subnet.
port	Gauge	port	port ID	Notification	Existence of port.
port.create	Delta	port	port ID	Notification	Creation requests for this port.
port.update	Delta	port	port ID	Notification	Update requests for this port.
router	Gauge	router	router ID	Notification	Existence of router.
router.create	Delta	router	router ID	Notification	Creation requests for this router.
router.update	Delta	router	router ID	Notification	Update requests for this router.
ip.floating	Gauge	ip	ip ID	Notification, Pollster	Existence of IP.

Name	Type	Unit	Resource	Origin	Note
ip.floating.create	Delta	ip	ip ID	Notification	Creation requests for this IP.
ip.floating.update	Delta	ip	ip ID	Notification	Update requests for this IP.
bandwidth	Delta	B	label ID	Notification	Bytes through this l3 metering label.

SDN controllers

The following meters are collected for SDN:

Table 8.16. SDN meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
switch	Gauge	switch	switch ID	Pollster	Existence of switch.
switch.port	Gauge	port	switch ID	Pollster	Existence of port.
switch.port.receive.packets	Cumulative	packet	switch ID	Pollster	Packets received on port.
switch.port.transmit.packets	Cumulative	packet	switch ID	Pollster	Packets transmitted on port.
switch.port.receive.bytes	Cumulative	B	switch ID	Pollster	Bytes received on port.
switch.port.transmit.bytes	Cumulative	B	switch ID	Pollster	Bytes transmitted on port.
switch.port.receive.drops	Cumulative	packet	switch ID	Pollster	Drops received on port.
switch.port.transmit.drops	Cumulative	packet	switch ID	Pollster	Drops transmitted on port.
switch.port.receive.errors	Cumulative	packet	switch ID	Pollster	Errors received on port.
switch.port.transmit.errors	Cumulative	packet	switch ID	Pollster	Errors transmitted on port.
switch.port.receive.frame_error	Cumulative	packet	switch ID	Pollster	Frame alignment errors received on port.
switch.port.receive.overrun_error	Cumulative	packet	switch ID	Pollster	Overrun errors received on port.
switch.port.receive.crc_error	Cumulative	packet	switch ID	Pollster	CRC errors received on port.
switch.port.collision.count	Cumulative	count	switch ID	Pollster	Collisions on port.
switch.table	Gauge	table	switch ID	Pollster	Duration of table.
switch.table.active.entries	Gauge	entry	switch ID	Pollster	Active entries in table.
switch.table.lookup.packets	Gauge	packet	switch ID	Pollster	Lookup packets for table.
switch.table.matched.packets	Gauge	packet	switch ID	Pollster	Packets matches for table.
switch.flow	Gauge	flow	switch ID	Pollster	Duration of flow.
switch.flow.duration.seconds	Gauge	s	switch ID	Pollster	Duration of flow in seconds.
switch.flow.duration.nanoseconds	Gauge	ns	switch ID	Pollster	Duration of flow in nanoseconds.
switch.flow.packets	Cumulative	packet	switch ID	Pollster	Packets received.
switch.flow.bytes	Cumulative	B	switch ID	Pollster	Bytes received.

These meters are available for OpenFlow based switches. In order to enable these meters, each driver needs to be properly configured.

LoadBalancer as a Service (LBaaS)

The following meters are collected for LBaaS:

Table 8.17. LoadBalancer as a Service meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					

Name	Type	Unit	Resource	Origin	Note
network.services.lb.pool	Gauge	pool	pool ID	Notification, Pollster	Existence of a LB pool.
network.services.lb.vip	Gauge	vip	vip ID	Notification, Pollster	Existence of a LB VIP.
network.services.lb.member	Gauge	member	member ID	Notification, Pollster	Existence of a LB member.
network.services.lb.health_monitor	Gauge	health_monitor	monitor ID	Notification, Pollster	Existence of a LB health probe.
network.services.lb.total.connections	Cumulative	connection	pool ID	Pollster	Total connections on a LB.
network.services.lb.active.connections	Gauge	connection	pool ID	Pollster	Active connections on a LB.
network.services.lb.incoming.bytes	Cumulative	B	pool ID	Pollster	Number of incoming Bytes.
network.services.lb.outgoing.bytes	Cumulative	B	pool ID	Pollster	Number of outgoing Bytes.
Meters added in the Kilo release					
network.services.lb.pool.create	Delta	pool	pool ID	Notification	LB pool was created.
network.services.lb.pool.update	Delta	pool	pool ID	Notification	LB pool was updated.
network.services.lb.vip.create	Delta	vip	vip ID	Notification	LB VIP was created.
network.services.lb.vip.update	Delta	vip	vip ID	Notification	LB VIP was updated.
network.services.lb.member.create	Delta	member	member ID	Notification	LB member was created.
network.services.lb.member.update	Delta	member	member ID	Notification	LB member was updated.
network.services.lb.health_monitor.create	Delta	health_monitor	monitor ID	Notification	LB health probe was created.
network.services.lb.health_monitor.update	Delta	health_monitor	monitor ID	Notification	LB health probe was updated.

VPN as a Service (VPNaaS)

The following meters are collected for VPNaaS:

Table 8.18. VPN as a Service meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					
network.services.vpn	Gauge	vpnservice	vpn ID	Notification, Pollster	Existence of a VPN.
network.services.vpn.connections	Gauge	ipsec_site_connection	connection ID	Notification, Pollster	Existence of an IPSec connection.
Meters added in the Kilo release					
network.services.vpn.create	Delta	vpnservice	vpn ID	Notification	VPN was created.
network.services.vpn.update	Delta	vpnservice	vpn ID	Notification	VPN was updated.
network.services.vpn.connections.create	Delta	ipsec_site_connection	connection ID	Notification	IPSec connection was created.
network.services.vpn.connections.update	Delta	ipsec_site_connection	connection ID	Notification	IPSec connection was updated.

Name	Type	Unit	Resource	Origin	Note
network.services.vpn.ipsecpolicy	Gauge	ipsecpolicy	ipsecpolicy ID	Notification, Pollster	Existence of an IPSec policy.
network.services.vpn.ipsecpolicy.create	Delta	ipsecpolicy	ipsecpolicy ID	Notification	IPSec policy was created.
network.services.vpn.ipsecpolicy.update	Delta	ipsecpolicy	ipsecpolicy ID	Notification	IPSec policy was updated.
network.services.vpn.ikepolicy	Gauge	ikepolicy	ikepolicy ID	Notification, Pollster	Existence of an Ike policy.
network.services.vpn.ikepolicy.create	Delta	ikepolicy	ikepolicy ID	Notification	Ike policy was created.
network.services.vpn.ikepolicy.update	Delta	ikepolicy	ikepolicy ID	Notification	Ike policy was updated.

Firewall as a Service (FWaaS)

The following meters are collected for FWaaS:

Table 8.19. Firewall as a Service meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					
network.services.firewall	Gauge	firewall	firewall ID	Notification, Pollster	Existence of a firewall.
network.services.firewall.policy	Gauge	firewall_policy	firewall ID	Notification, Pollster	Existence of a firewall policy.
Meters added in the Kilo release					
network.services.firewall.create	Delta	firewall	firewall ID	Notification	Firewall was created.
network.services.firewall.update	Delta	firewall	firewall ID	Notification	Firewall was updated.
network.services.firewall.policy.create	Delta	firewall_policy	policy ID	Notification	Firewall policy was created.
network.services.firewall.policy.update	Delta	firewall_policy	policy ID	Notification	Firewall policy was updated.
network.services.firewall.rule	Gauge	firewall_rule	rule ID	Notification	Existence of a firewall rule.
network.services.firewall.rule.create	Delta	firewall_rule	rule ID	Notification	Firewall rule was created.
network.services.firewall.rule.update	Delta	firewall_rule	rule ID	Notification	Firewall rule was updated.

Orchestration module

The following meters are collected for the Orchestration module:

Table 8.20. Metrics for the Orchestration module

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
stack.create	Delta	stack	stack ID	Notification	Stack was successfully created.
stack.update	Delta	stack	stack ID	Notification	Stack was successfully updated.
stack.delete	Delta	stack	stack ID	Notification	Stack was successfully deleted.

Name	Type	Unit	Resource	Origin	Note
stack.resume	Delta	stack	stack ID	Notification	Stack was successfully resumed.
stack.suspend	Delta	stack	stack ID	Notification	Stack was successfully suspended.

Data processing service for OpenStack

The following meters are collected for the Data processing service for OpenStack:

Table 8.21. Metrics of the Data processing service for OpenStack

Name	Type	Unit	Resource	Origin	Note
Meters added in the Juno release					
cluster.create	Delta	cluster	cluster ID	Notification	Cluster was successfully created.
cluster.update	Delta	cluster	cluster ID	Notification	Cluster was successfully updated.
cluster.delete	Delta	cluster	cluster ID	Notification	Cluster was successfully deleted.

Key Value Store module

The following meters are collected for the Key Value Store module:

Table 8.22. Key Value Store module meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Kilo release					
magnetodb.table.create	Gauge	table	table ID	Notification	Table was successfully created.
magnetodb.table.delete	Gauge	table	table ID	Notification	Table was successfully deleted.
magnetodb.table.index.count	Gauge	index	table ID	Notification	Number of indices created in a table.

Energy

The following energy related meters are available:

Table 8.23. Energy meters

Name	Type	Unit	Resource	Origin	Note
Meters added in the Icehouse release or earlier					
energy	Cumulative	kWh	probe ID	Pollster	Amount of energy.
power	Gauge	W	probe ID	Pollster	Power consumption.

Events

In addition to meters, the Telemetry module collects events triggered within an OpenStack environment. This section provides a brief summary of the events format in the Telemetry module.

While a sample represents a single, numeric datapoint within a time-series, an event is a broader concept that represents the state of a resource at a point in time. The state may be described using various data types including non-numeric data such as an instance's flavor. In general, events represent any action made in the OpenStack system.

Event configuration

To enable the creation and storage of events in the Telemetry module `store_events` option needs to be set to `True`. For further configuration options, see the event section in the *OpenStack Configuration Reference*.

Note

It is advisable to set `disable_non_metric_meters` to `True` when enabling events in the Telemetry module. The Telemetry module historically represented events as metering data, which may create duplication of data if both events and non-metric meters are enabled.

Event structure

Events captured by the Telemetry module are represented by five key attributes:

event_type	A dotted string defining what event occurred such as `compute.instance.resize.start"`.
message_id	A UUID for the event.
generated	A timestamp of when the event occurred in the system.
traits	A flat mapping of key-value pairs which describe the event. The event's Traits contain most of the details of the event. Traits are typed, and can be strings, integers, floats, or datetimes.
raw	Mainly for auditing purpose, the full event message can be stored (unindexed) for future evaluation.

Event indexing

The general philosophy of notifications in OpenStack is to emit any and all data someone might need, and let the consumer filter out what they are not interested in. In order to make processing simpler and more efficient, the notifications are stored and processed within Ceilometer as events. The notification payload, which can be an arbitrarily complex JSON data structure, is converted to a flat set of key-value pairs. This conversion is specified by a config file.

Note

The event format is meant for efficient processing and querying. Storage of complete notifications for auditing purposes can be enabled by configuring `store_raw` option.

Event conversion

The conversion from notifications to events is driven by a configuration file defined by the `definitions_cfg_file` in the `ceilometer.conf` configuration file.

This includes descriptions of how to map fields in the notification body to Traits, and optional plug-ins for doing any programmatic translations (splitting a string, forcing case).

The mapping of notifications to events is defined per event_type, which can be wildcarded. Traits are added to events if the corresponding fields in the notification exist and are non-null.

Note

The default definition file included with the Telemetry module contains a list of known notifications and useful traits. The mappings provided can be modified to include more or less data according to user requirements.

If the definitions file is not present, a warning will be logged, but an empty set of definitions will be assumed. By default, any notifications that do not have a corresponding event definition in the definitions file will be converted to events with a set of minimal traits. This can be changed by setting the option `drop_unmatched_notifications` in the `ceilometer.conf` file. If this is set to True, any unmapped notifications will be dropped.

The basic set of traits (all are TEXT type) that will be added to all events if the notification has the relevant data are: service (notification's publisher), tenant_id, and request_id. These do not have to be specified in the event definition, they are automatically added, but their definitions can be overridden for a given event_type.

Event definitions format

The event definitions file is in YAML format. It consists of a list of event definitions, which are mappings. Order is significant, the list of definitions is scanned in reverse order to find a definition which matches the notification's event_type. That definition will be used to generate the event. The reverse ordering is done because it is common to want to have a more general wildcarded definition (such as `compute.instance.*`) with a set of traits common to all of those events, with a few more specific event definitions afterwards that have all of the above traits, plus a few more.

Each event definition is a mapping with two keys:

event_type	This is a list (or a string, which will be taken as a 1 element list) of event_types this definition will handle. These can be wildcarded with unix shell glob syntax. An exclusion listing (starting with a `!`) will exclude any types listed from matching. If only exclusions are listed, the definition will match anything not matching the exclusions.
traits	This is a mapping, the keys are the trait names, and the values are trait definitions.

Each trait definition is a mapping with the following keys:

fields	A path specification for the field(s) in the notification you wish to extract for this trait. Specifications can be written to match multiple possible fields. By default the value will be the first such field. The paths can be specified with a dot syntax (`payload.host`). Square bracket syntax (`payload[host]`) is also supported. In either case, if the key for the field you are looking for contains special characters, like `.`, it will need to be quoted (with double or single quotes): `payload.image_meta.'org.openstack__1__architecture'`. The syntax used for the field specification is a variant of JSONPath

type	(Optional) The data type for this trait. Valid options are: `text`, `int`, `float`, and `datetime`. Defaults to `text` if not specified.
plugin	(Optional) Used to execute simple programmatic conversions on the value in a notification field.

Troubleshoot Telemetry

Logging in Telemetry

The Telemetry module has similar log settings as the other OpenStack services. Multiple options are available to change the target of logging, the format of the log entries and the log levels.

The log settings can be changed in `ceilometer.conf`. The list of configuration options are listed in the logging configuration options table in the Telemetry section in the *OpenStack Configuration Reference*.

By default `stderr` is used as standard output for the log messages. It can be changed to either a log file or syslog. The `debug` and `verbose` options are also set to false in the default settings, the default log levels of the corresponding modules can be found in the table referred above.

Recommended order of starting services

As it can be seen in Bug 1355809, the wrong ordering of service startup can result in data loss.

When the services are started for the first time or in line with the message queue service restart, it takes time while the `ceilometer-collector` services establishes the connection and joins or rejoins to the configured exchanges. Therefore, if the `ceilometer-agent-compute`, `ceilometer-agent-central` and the `ceilometer-agent-notification` services are started before `ceilometer-collector`, it can loose some messages sent by these services, while connecting to the message queue service.

The possibility of this issue to happen is higher, when the polling interval is set to a relatively short period. In order to avoid this situation, the recommended order of service startup is to start or restart the `ceilometer-collector` service after the message queue. All the other Telemetry services should be started or restarted after and the `ceilometer-agent-compute` should be the last in the sequence, as this component emits metering messages in order to send the samples to the collector.

Notification agent

In the Icehouse release of OpenStack a new service was introduced to be responsible for consuming notifications that are coming from other OpenStack services.

If the `ceilometer-agent-notification` service is not installed and started, samples originating from notifications will not be generated. In case of the lack of notification based samples, the state of this service and the log file of Telemetry should be checked first.

For the list of meters that are originated from notifications, see the Telemetry Measurements Reference.

Recommended `auth_url` to be used

When using the commandline client of Telemetry, the credentials and the `os_auth_url` has to be set in order to provide the possibility to the client to authenticate against OpenStack Identity. For further details about the credentials that has to be provided see the Python API reference of Telemetry.

The service catalog provided by OpenStack Identity contains the available URLs that are available for authentication. The URLs contain different `ports`, based on that the type of the given URL is `public`, `internal` or `admin`.

OpenStack Identity is about to change API version from v2 to v3. The `adminURL` endpoint (which is available via the port: `35357`) supports only the v3 version, while the other two supports both.

The commandline client of Telemetry is not adapted to the v3 version of the OpenStack Identity API. If the `adminURL` is used as `os_auth_url`, the **ceilometer** command results in the following error message:

```
$ ceilometer meter-list
Unable to determine the Keystone version to authenticate with using the given
auth_url: http://10.0.2.15:35357/v2.0
```

Therefore when specifying the `os_auth_url` parameter on the command line or by using environment variable, use the `internalURL` or `publicURL`.

For more details check the bug report Bug 1351841.

Telemetry best practices

The following are some suggested best practices to follow when deploying and configuring the Telemetry service. The best practices are divided into data collection and storage.

Data collection

1. The Telemetry module collects a continuously growing set of data. Not all the data will be relevant for a cloud administrator to monitor.

   - Based on your needs, you can edit the `pipeline.yaml` configuration file to include a selected number of meters while disregarding the rest.

   - By default, Telemetry service polls the service APIs every 10 minutes. You can change the polling interval on a per meter basis by editing the `pipeline.yaml` configuration file.

   Warning

   If the polling interval is too short, it will likely cause increase of stored data and the stress on the service APIs.

- Expand the configuration to have greater control over different meter intervals.

> **Note**
>
> For more information, see the Pipeline Configuration Options.

2. If you are using the Kilo version of Telemetry, you can delay or adjust polling requests by enabling the jitter support. This adds a random delay on how the polling agents send requests to the service APIs. To enable jitter, set `shuffle_time_before_polling_task` in the `ceilometer.conf` configuration file to an integer greater than 0.

3. If you are using Juno or later releases, based on the number of resources that will be polled, you can add additional central and compute agents as necessary. The agents are designed to scale horizontally.

> **Note**
>
> For more information see, HA deployment for Central and Compute Agents.

4. If you are using Juno or later releases, use the `notifier://` publisher rather than `rpc://` as there is a certain level of overhead that comes with RPC.

> **Note**
>
> For more information on RPC overhead, see RPC overhead info.

Data storage

1. We recommend that you avoid open-ended queries. In order to get better performance you can use reasonable time ranges and/or other query constraints for retrieving measurements.

   For example, this open-ended query might return an unpredictable amount of data:

```
$ceilometer sample-list --meter cpu -q 'resource_id=INSTANCE_ID_1'
```

   Whereas, this well-formed query returns a more reasonable amount of data, hence better performance:

```
$ceilometer sample-list --meter cpu -q 'resource_id=
INSTANCE_ID_1;timestamp>2015-05-01T00:00:00;timestamp<2015-06-01T00:00:00'
```

2. You can install the API behind `mod_wsgi`, as it provides more settings to tweak, like `threads` and `processes` in case of `WSGIDaemon`.

> **Note**
>
> For more information on how to configure `mod_wsgi`, see the Install Documentation.

3. The collection service provided by the Telemetry project is not intended to be an archival service. Set a Time to Live (TTL) value to expire data and minimize the database size. If you would like to keep your data for longer time period, you may consider storing it in a data warehouse outside of Telemetry.

   Note

   For more information on how to set the TTL, see TTL support for various back ends.

4. We recommend that you do not use SQLAlchemy back end prior to the Juno release, as it previously contained extraneous relationships to handle deprecated data models. This resulted in extremely poor query performance.

5. We recommend that you do not run MongoDB on the same node as the controller. Keep it on a separate node optimized for fast storage for better performance. Also it is advisable for the MongoDB node to have a lot of memory.

   Note

   For more information on how much memory you need, see MongoDB FAQ.

6. Use replica sets in MongoDB. Replica sets provide high availability through automatic failover. If your primary node fails, MongoDB will elect a secondary node to replace the primary node, and your cluster will remain functional.

   For more information on replica sets, see the MongoDB replica sets docs.

7. Use sharding in MongoDB. Sharding helps in storing data records across multiple machines and is the MongoDB's approach to meet the demands of data growth.

   For more information on sharding, see the MongoDB sharding docs.

9. Database

Table of Contents

The Database service module provides database management features.

Introduction

The Database service provides scalable and reliable cloud provisioning functionality for both relational and non-relational database engines. Users can quickly and easily use database features without the burden of handling complex administrative tasks. Cloud users and database administrators can provision and manage multiple database instances as needed.

The Database service provides resource isolation at high performance levels, and automates complex administrative tasks such as deployment, configuration, patching, backups, restores, and monitoring.

Create a datastore

An administrative user can create datastores for a variety of databases.

This section assumes you do not yet have a MySQL datastore, and shows you how to create a MySQL datastore and populate it with a MySQL 5.5 datastore version.

To create a datastore

1. **Create a trove image**

   Create an image for the type of database you want to use, for example, MySQL, MongoDB, Cassandra, and so on.

   This image must have the trove guest agent installed, and it must have the `trove-guestagent.conf` file configured to connect to your OpenStack environment. To configure `trove-guestagent.conf`, add the following lines to `trove-guestagent.conf` on the guest instance you are using to build your image:

   ```
   rabbit_host = controller
   rabbit_password = RABBIT_PASS
   nova_proxy_admin_user = admin
   nova_proxy_admin_pass = ADMIN_PASS
   nova_proxy_admin_tenant_name = service
   trove_auth_url = http://controller:35357/v2.0
   ```

   This example assumes you have created a MySQL 5.5 image called `mysql-5.5.qcow2`.

2. **Register image with Image service**

   You need to register your guest image with the Image service.

   In this example, you use the glance **image-create** command to register a `mysql-5.5.qcow2` image.

```
$ glance image-create --name mysql-5.5 --disk-format qcow2 --container-
format bare --is-public True < mysql-5.5.qcow2
+------------------+--------------------------------------+
| Property         | Value                                |
+------------------+--------------------------------------+
| checksum         | d41d8cd98f00b204e9800998ecf8427e     |
| container_format | bare                                 |
| created_at       | 2014-05-23T21:01:18                  |
| deleted          | False                                |
| deleted_at       | None                                 |
| disk_format      | qcow2                                |
| id               | bb75f870-0c33-4907-8467-1367f8cb15b6 |
| is_public        | True                                 |
| min_disk         | 0                                    |
| min_ram          | 0                                    |
| name             | mysql-5.5                            |
| owner            | 1448da1223124bb291f5ae8e9af4270d     |
| protected        | False                                |
| size             | 0                                    |
| status           | active                               |
| updated_at       | 2014-05-23T21:01:22                  |
| virtual_size     | None                                 |
+------------------+--------------------------------------+
```

3. **Create the datastore**

   Create the datastore that will house the new image. To do this, use the trove-manage **datastore_update** command.

   This example uses the following arguments:

Argument	Description	In this example:
config file	The configuration file to use.	`--config-file=/etc/trove/trove.conf`
name	Name you want to use for this datastore.	`mysql`
default version	You can attach multiple versions/images to a datastore. For example, you might have a MySQL 5.5 version and a MySQL 5.6 version. You can designate one version as the default, which the system uses if a user does not explicitly request a specific version.	`""` At this point, you do not yet have a default version, so pass in an empty string.

   Example:

```
$ trove-manage --config-file=/etc/trove/trove.conf datastore_update mysql
 ""
```

4. **Add a version to the new datastore**

   Now that you have a `mysql` datastore, you can add a version to it, using the trove-manage **datastore_version_update** command. The version indicates which guest image to use.

   This example uses the following arguments:

Argument	Description	In this example:
config file	The configuration file to use.	`--config-file=/etc/trove/trove.conf`
datastore	The name of the datastore you just created via trove-manage **datastore_update**.	`mysql`
version name	The name of the version you are adding to the datastore.	`mysql-5.5`
datastore manager	Which datastore manager to use for this version. Typically, the datastore manager is identified by one of the following strings, depending on the database: • mysql • redis • mongodb • cassandra • couchbase • percona	`mysql`
glance ID	The ID of the guest image you just added to the Identity Service. You can get this ID by using the glance **image-show** *`IMAGE_NAME`* command.	`bb75f870-0c33-4907-8467-1367f8cb15b6`
packages	If you want to put additional packages on each guest that you create with this datastore version, you can list the package names here.	`""` In this example, the guest image already contains all the required packages, so leave this argument empty.
active	Set this to either 1 or 0: • `1` = active • `0` = disabled	`1`

   Example:

```
$ trove-manage --config-file=/etc/trove/trove.conf
 datastore_version_update \
  mysql mysql-5.5 mysql GLANCE_ID "" 1
```

   Optional. Set your new version as the default version. To do this, use the trove-manage **datastore_update** command again, this time specifying the version you just created.

```
$ trove-manage --config-file=/etc/trove/trove.conf datastore_update mysql
 "mysql-5.5"
```

5. **Load validation rules for configuration groups**

Applies only to MySQL and Percona datastores

- If you just created a MySQL or Percona datastore, then you need to load the appropriate validation rules, as described in this step.
- If you just created a different datastore, skip this step.

Background. You can manage database configuration tasks by using configuration groups. Configuration groups let you set configuration parameters, in bulk, on one or more databases.

When you set up a configuration group using the trove **configuration-create** command, this command compares the configuration values you are setting against a list of valid configuration values that are stored in the `validation-rules.json` file.

Operating System	Location of `validation-rules.json`	Notes
Ubuntu 14.04	`/usr/lib/python2.7/dist-packages/trove/templates/`*DATASTORE_NAME*	*DATASTORE_NAME* is the name of either the MySQL datastore or the Percona datastore. This is typically either `mysql` or `percona`.
RHEL 7, CentOS 7, Fedora 20, and Fedora 21	`/usr/lib/python2.7/site-packages/trove/templates/`*DATASTORE_NAME*	*DATASTORE_NAME* is the name of either the MySQL datastore or the Percona datastore. This is typically either `mysql` or `percona`.

Therefore, as part of creating a datastore, you need to load the `validation-rules.json` file, using the trove-manage **db_load_datastore_config_parameters** command. This command takes the following arguments:

- Datastore name
- Datastore version
- Full path to the `validation-rules.json` file

This example loads the `validation-rules.json` file for a MySQL database on Ubuntu 14.04:

```
$ trove-manage db_load_datastore_config_parameters mysql "mysql-5.5" \
/usr/lib/python2.7/dist-packages/trove/templates/mysql/validation-rules.
json
```

6. **Validate datastore**

To validate your new datastore and version, start by listing the datastores on your system:

```
$ trove datastore-list
```

```
+--------------------------------------+--------------+
|                  id                  |     name     |
+--------------------------------------+--------------+
| 10000000-0000-0000-0000-000000000001 | Legacy MySQL |
| e5dc1da3-f080-4589-a4c2-eff7928f969a |    mysql     |
+--------------------------------------+--------------+
```

Take the ID of the mysql datastore and pass it in with the **datastore-version-list** command:

```
$ trove datastore-version-list DATASTORE_ID
+--------------------------------------+-----------+
|                  id                  |    name   |
+--------------------------------------+-----------+
| 36a6306b-efd8-4d83-9b75-8b30dd756381 | mysql-5.5 |
+--------------------------------------+-----------+
```

Configure a cluster

An administrative user can configure various characteristics of a MongoDB cluster.

Query routers and config servers

Background. Each cluster includes at least one query router and one config server. Query routers and config servers count against your quota. When you delete a cluster, the system deletes the associated query router(s) and config server(s).

Configuration. By default, the system creates one query router and one config server per cluster. You can change this by editing the /etc/trove/trove.conf file. These settings are in the [mongodb] section of the file:

Setting	Valid values are:
num_config_servers_per_cluster	1 or 3
num_query_routers_per_cluster	1 or 3

10. Orchestration

Table of Contents

Orchestration is an orchestration engine that provides the possibility to launch multiple composite cloud applications based on templates in the form of text files that can be treated like code. A native Heat Orchestration Template (HOT) format is evolving, but it also endeavors to provide compatibility with the AWS CloudFormation template format, so that many existing CloudFormation templates can be launched on OpenStack.

Introduction

Orchestration is a tool for orchestrating clouds that automatically configures and deploys resources in stacks. Such deployments can be simple — like deploying WordPress on Ubuntu with a SQL back end. And they can be quite complex, like launching a group of servers that autoscale: starting and stopping based on realtime CPU loading information from the Telemetry module.

Orchestration stacks are defined with templates, which are non-procedural documents describing tasks in terms of resources, parameters, inputs, constraints and dependencies. When Orchestration module was originally introduced, it worked with AWS CloudFormation templates, which are in JSON format.

Now, Orchestration also executes HOT (Heat Orchestration Template) templates, written in YAML: a terse notation that loosely follows Python/Ruby-type structural conventions (colons, returns, indentation) so it's more easily to write, parse, grep, generate with tools, and maintain with source-code management systems.

Orchestration can be accessed via the CLI, and using RESTful queries. Orchestration module provides both an OpenStack-native REST API and a CloudFormation-compatible Query API. Orchestration is also integrated with OpenStack dashboard in order to launching stacks from templates through web-interface.

For more details how to use Orchestration module command-line see OpenStack Command line interface reference

Orchestration authorization model

Orchestration authorization model defines the process of authorization that orchestration module uses to authorize requests during so called deferred operations. The typical example of such operation is autoscaling group update when heat requests another components (nova, neutron or others) to extend (reduce) capacity of autoscaling group.

At the current moment, Orchestration provides two kinds of authorization models:

- Password authorization.
- Authorization with OpenStack Identity trusts.

Password authorization

Password authorization is the initial authorization model that was supported by Orchestration module. This kind of authorization requires from a user to pass a password to Orchestration. Orchestration stores the encrypted password in database and uses it for deferred operations.

The following steps are executed for password authorization:

1. User requests stack creation, providing a token and username/password (python-heatclient or OpenStack dashboard normally requests the token for you).
2. If the stack contains any resources marked as requiring deferred operations orchestration engine will fail validation checks if no username/password is provided.
3. The username/password are encrypted and stored in the orchestration DB.
4. Stack creation is completed.
5. At some later stage Orchestration retrieves the credentials and requests another token on behalf of the user, the token is not limited in scope and provides access to all roles of the stack owner.

Keystone trusts authorization

OpenStack Identity trusts is the new authorization method available since IceHouse release.

Trusts are an OpenStack Identity extension, which provide a method to enable delegation, and optionally impersonation via OpenStack Identity. The key terminology is *trustor* (the user delegating) and *trustee* (the user being delegated to).

To create a trust, the *trustor*(in this case the user creating the stack in Orchestration module) provides OpenStack Identity with the following information:

- The ID of the *trustee*(who you want to delegate to, in this case the Orchestration service user).
- The roles to be delegated(configurable via the `heat.conf`, but it needs to contain whatever roles are required to perform the deferred operations on the users behalf, e.g launching a OpenStack Compute instance in response to an AutoScaling event).
- Whether to enable impersonation.

OpenStack Identity then provides a trust_id, which can be consumed by the trustee (and *only* the trustee) to obtain a *trust scoped token*. This token is limited in scope such that the trustee has limited access to those roles delegated, along with effective impersonation of the trustor user, if it was selected when creating the trust. More information is available in Identity management section.

The following steps are executed for trusts authorization:

1. User creates a stack via an API request (only the token is required).

2. Orchestration uses the token to create a trust between the stack owner (trustor) and the heat service user (trustee), delegating a special role (or roles) as defined in the *trusts_delegated_roles* list in the heat configuration file. By default heat sets all roles from trustor available for trustee. Deployers may modify this list to reflect local RBAC policy, e.g to ensure the heat process can only access those services expected while impersonating a stack owner.

3. Orchestration stores the encrypted *trust id* in the Orchestration DB.

4. When a deferred operation is required, Orchestration retrieves the *trust id*, and requests a trust scoped token which enables the service user to impersonate the stack owner for the duration of the deferred operation, e.g to launch some OpenStack Compute instances on behalf of the stack owner in response to an AutoScaling event.

Authorization model configuration

Password authorization model had been the default authorization model enabled for Orchestration module before Kilo release. Since Kilo release trusts authorization model has been enabled by default.

To enable password authorization model the following change should be made in `heat.conf`:

```
deferred_auth_method=password
```

To enable trusts authorization model the following change should be made in `heat.conf`:

```
deferred_auth_method=trusts
```

To specify trustor roles that will be delegated to trustee during authorization `trusts_delegated_roles` parameter should be specified in `heat.conf`. If `trusts_delegated_roles` is not defined then all trustor roles will be delegated to trustee. Please pay attention that trust delegated roles should be pre-configured in OpenStack Identity before using it in Orchestration module.

Stack domain users

Orchestration stack domain users allows heat to authorize inside VMs booted and execute the following operations:

- Provide metadata to agents inside instances, which poll for changes and apply the configuration expressed in the metadata to the instance.

- Detect signal completion of some action, typically configuration of software on a VM after it is booted (because OpenStack Compute moves the state of a VM to "Active" as soon as it spawns it, not when orchestration has fully configured it).

- Provide application level status or meters from inside the instance. For example, allow AutoScaling actions to be performed in response to some measure of performance or quality of service.

Orchestration provides API's which enable all of these things, but all of those API's require some sort of authentication. For example, credentials to access the instance agent is running on. The heat-cfntools agents use signed requests, which requires an ec2 keypair created via OpenStack Identity, which is then used to sign requests to the Orchestration cloudformation and cloudwatch compatible API's, which are authenticated by Orchestration via signature validation (which uses the OpenStack Identity ec2tokens extension). Stack domain users allow to encapsulate all stack-defined users (users created as a result of things contained in a Orchestration template) in a separate domain, which is created specifically to contain things related only to Orchestration stacks. A user is created which is the *domain admin*, and Orchestration uses that user to manage the lifecycle of the users in the *stack user domain*.

Stack domain users configuration

To configure stack domain users the following steps shall be executed:

1. A special OpenStack Identity service domain is created. For example, the one called `heat` and the ID is set in the `stack_user_domain` option in `heat.conf`.

2. A user with sufficient permissions to create and delete projects and users in the `heat` domain is created.

3. The username and password for the domain admin user is set in `heat.conf` (`stack_domain_admin` and `stack_domain_admin_password`). This user administers *stack domain users* on behalf of stack owners, so they no longer need to be admins themselves, and the risk of this escalation path is limited because the `heat_domain_admin` is only given administrative permission for the `heat` domain.

You must complete the following steps to setup stack domain users:

1. Create the domain:

   `$OS_TOKEN` refers to a token. For example, the service admin token or some other valid token for a user with sufficient roles to create users and domains. `$KS_ENDPOINT_V3` refers to the v3 OpenStack Identity endpoint (for example `http://`*`keystone_address`*`:5000/v3` where *keystone_address* is the IP address or resolvable name for the OpenStack Identity service).

   ```
   $ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-identity-api-
   version=3 domain create heat --description "Owns users and projects created
    by heat"
   ```

   The domain ID is returned by this command, and is referred to as `$HEAT_DOMAIN_ID below`.

   The domain ID is returned by this command, and is referred to as `$HEAT_DOMAIN_ID` below.

2. Create the user:

   ```
   $ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-identity-
   api-version=3 user create --password $PASSWORD --domain $HEAT_DOMAIN_ID
    heat_domain_admin --description "Manages users and projects created by
    heat"
   ```

The user ID is returned by this command and is referred to as `$DOMAIN_ADMIN_ID` below.

3. Make the user a domain admin:

```
$ openstack --os-token $OS_TOKEN --os-url=$KS_ENDPOINT_V3 --os-identity-api-
version=3 role add --user $DOMAIN_ADMIN_ID --domain $HEAT_DOMAIN_ID admin
```

Then you need to add the domain ID, username and password from these steps to `heat.conf`:

```
stack_domain_admin_password = password
stack_domain_admin = heat_domain_admin
stack_user_domain = domain id returned from domain create above
```

Usage workflow

The following steps will be executed during stack creation:

1. Orchestration creates a new "stack domain project" in the "heat" domain, if the stack contains any resources which require creation of a "stack domain user".

2. Any resources which require a user, Orchestration creates the user in the "stack domain project", which is associated with the heat stack in the heat database, but is completely separate and unrelated (from an authentication perspective) to the stack owners project (the users created in the stack domain are still assigned the `heat_stack_user` role, so the API surface they can access is limited via policy.json. See OpenStack Identity documentation for more info).

3. When API requests are processed, Heat Orchestration does an internal lookup, and allow stack details for a given stack to be retrieved from the database for both the stack owner's project (the default API path to the stack), and also the *stack domain project*, subject to the policy.json restrictions.

To clarify that last point, that means there are now two paths which can result in retrieval of the same data via the Orchestration API. The example for resource-metadata is below:

```
GET v1/{stack_owner_project_id}/stacks/{stack_name}/{stack_id}/resources/
{resource_name}/metadata
```

or:

```
GET v1/{stack_domain_project_id}/stacks/{stack_name}/{stack_id}/resources/
{resource_name}/metadata
```

The stack owner would use the former (via `heat resource-metadata {stack_name} {resource_name}`), and any agents in the instance will use the latter.

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 openSUSE 13.2 and SUSE Linux Enterprise Server 12*
- *Installation Guide for Red Hat Enterprise Linux 7, CentOS 7, and Fedora 21*
- *Installation Guide for Ubuntu 14.04 (LTS)*

The following books explain how to configure and run an OpenStack cloud:

- *Architecture Design Guide*
- *Cloud Administrator Guide*
- *Configuration Reference*
- *Operations Guide*
- *Networking 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)

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 https://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 https://wiki.openstack.org/wiki/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 OpenStack Compute, 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 service (ironic)
- Bugs: Data processing service (sahara)
- Bugs: Database service (trove)
- Bugs: Orchestration (heat)
- Bugs: Telemetry (ceilometer)
- Bugs: Message Service (zaqar)
- Bugs: OpenStack API Documentation (developer.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 https://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:** https://wiki.debian.org/OpenStack
- **CentOS, Fedora, and Red Hat Enterprise Linux:** https://www.rdoproject.org/
- **openSUSE and SUSE Linux Enterprise Server:** https://en.opensuse.org/Portal:OpenStack
- **Ubuntu:** https://wiki.ubuntu.com/ServerTeam/CloudArchive