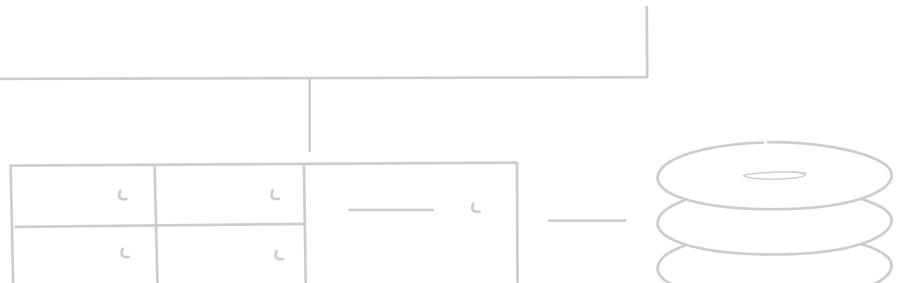
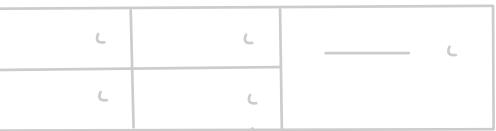


# OpenStack Compute

## Administration Guide

Grizzly, 2013.1 (Jun 5, 2013)



## OpenStack Compute Administration Guide

Grizzly, 2013.1 (2013-06-05)

Copyright © 2010-2013 OpenStack Foundation Some rights reserved.

OpenStack™ Compute offers open source software for cloud administration and management for any organization. This manual provides guidance for installing, managing, and understanding the software that runs OpenStack Compute.

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

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

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



Except where otherwise noted, this document is licensed under  
**Creative Commons Attribution ShareAlike 3.0 License.**  
<http://creativecommons.org/licenses/by-sa/3.0/legalcode>

## List of Figures

2.1. Base image state with no running instances .....	14
2.2. Instance creation from image and run time state .....	15
2.3. End state of image and volume after instance exits .....	15
4.1. KVM, Flat, MySQL, and Glance, OpenStack or EC2 API .....	33
4.2. KVM, Flat, MySQL, and Glance, OpenStack or EC2 API .....	34
4.3. MooseFS deployment for OpenStack .....	48
10.1. Flat network, all-in-one server installation .....	210
10.2. Flat network, single interface, multiple servers .....	211
10.3. Flat network, multiple interfaces, multiple servers .....	211
10.4. Flat DHCP network, multiple interfaces, multiple servers with libvirt driver .....	213
10.5. Flat DHCP network, multiple interfaces, multiple servers, network HA with XenAPI driver .....	214
10.6. Single adaptor hosts, first route .....	215
10.7. Single adaptor hosts, second route .....	216
10.8. VLAN network, multiple interfaces, multiple servers, network HA with XenAPI driver .....	220
10.9. Configuring Viscosity .....	231
10.10. multinic flat manager .....	239
10.11. multinic flatdhcp manager .....	240
10.12. multinic VLAN manager .....	241
10.13. High Availability Networking Option .....	243
12.1. Filtering .....	251
12.2. Computing weighted costs .....	257
15.1. NoVNC Process .....	295

# List of Tables

1.1. Types of Storage .....	10
3.1. Hardware Recommendations .....	18
4.1. Description of nova.conf log file configuration options .....	34
4.2. Description of nova.conf file configuration options for hypervisors .....	35
4.3. Description of nova.conf configuration options for authentication .....	38
4.4. Description of nova.conf file configuration options for credentials (crypto) .....	39
4.5. Description of nova.conf file configuration options for LDAP .....	39
4.6. Description of nova.conf configuration options for IPv6 .....	41
4.7. Description of nova.conf file configuration options for S3 access to image storage .....	41
4.8. Description of nova.conf file configuration options for live migration .....	45
4.9. Description of nova.conf configuration options for databases .....	53
4.10. Description of nova .conf configuration options for Remote Procedure Calls and RabbitMQ Messaging .....	54
4.11. Description of nova .conf configuration options for Tuning RabbitMQ Messaging .....	54
4.12. Remaining nova .conf configuration options for Qpid support .....	55
4.13. Description of nova .conf configuration options for Customizing Exchange or Topic Names .....	56
4.14. Description of nova.conf API related configuration options .....	56
4.15. Default API Rate Limits .....	57
4.16. Description of nova.conf file configuration options for EC2 API .....	58
5.1. Description of common nova .conf configuration options for the Compute API, RabbitMQ, EC2 API, S3 API, instance types .....	62
5.2. Description of nova.conf configuration options for databases .....	66
5.3. Description of nova.conf configuration options for IPv6 .....	67
5.4. Description of nova.conf log file configuration options .....	67
5.5. Description of nova.conf file configuration options for nova- services .....	68
5.6. Description of nova.conf file configuration options for credentials (crypto) .....	69
5.7. Description of nova.conf file configuration options for policies (policy.json) .....	69
5.8. Description of nova.conf file configuration options for quotas .....	69
5.9. Description of nova.conf file configuration options for testing purposes .....	70
5.10. Description of nova.conf configuration options for authentication .....	70
5.11. Description of nova.conf configuration options for LDAP .....	71
5.12. Description of nova.conf file configuration options for roles and authentication .....	72
5.13. Description of nova.conf file configuration options for EC2 API .....	72
5.14. Description of nova.conf file configuration options for VNC access to guest instances .....	73
5.15. Description of nova.conf [spice] section configuration options for SPICE HTML5 access to guest instances .....	73
5.16. Description of nova.conf file configuration options for networking options .....	73
5.17. Description of nova.conf file configuration options for live migration .....	75
5.18. Description of nova.conf file configuration options for compute nodes .....	76
5.19. Description of nova.conf file configuration options for bare metal deployment....	76
5.20. Description of nova.conf file configuration options for hypervisors .....	77
5.21. Description of nova.conf file configuration options for console access to VMs on VMWare VMRC or XenAPI .....	80

---

5.22. Description of nova.conf file configuration options for S3 access to image storage .....	80
5.23. Description of nova.conf file configuration options for schedulers that use algorithms to assign VM launch on particular compute hosts .....	81
5.24. Description of nova.conf file configuration options for config drive features .....	82
5.25. Description of nova.conf file configuration options for volumes attached to VMs .....	82
6.1. Description of keystone.conf file configuration options for LDAP .....	111
8.1. List of configuration flags for NFS .....	151
9.1. Description of nova.conf file configuration options for hypervisors .....	174
14.1. Description of rootwrap.conf configuration options .....	274
14.2. Description of rootwrap.conf configuration options .....	274
15.1. Description of nova.conf file configuration options for VNC access to guest instances .....	296
15.2. Description of nova.conf [spice] section configuration options for SPICE HTML5 access to guest instances .....	299
17.1. OSes supported .....	305

# Preface

## Table of Contents

Document Change History ..... 6

OpenStack™ Compute offers open source software for cloud administration and management for any organization. This manual provides guidance for installing, managing, and understanding the software that runs OpenStack Compute.

## Document Change History

The most recent changes are described in the table below:

Revision Date	Summary of Changes
May 9, 2013	<ul style="list-style-type: none"><li>Updated the book title for consistency.</li></ul>
Apr 30, 2013	<ul style="list-style-type: none"><li>Release for Grizzly, 2013.1.</li></ul>
Apr 11, 2013	<ul style="list-style-type: none"><li>Removes nova-manage commands when possible.</li></ul>
Mar 25, 2013	<ul style="list-style-type: none"><li>Moves block storage into separate administration manual.</li></ul>
Feb 19, 2013	<ul style="list-style-type: none"><li>Adds SPICE HTML5 information for remote access to server instances.</li></ul>
Nov 9, 2012	<ul style="list-style-type: none"><li>Folsom release of this document.</li><li>Adds Block Storage (Cinder) Volume service configuration and troubleshooting information.</li></ul>
Sep 18, 2012	<ul style="list-style-type: none"><li>Adds Hyper-V configuration information.</li></ul>
Sep 17, 2012	<ul style="list-style-type: none"><li>Adds updates to filter information, networking documentation, storage sections, nova-manage, configuration options and other various doc improvements.</li></ul>
Aug 1, 2012	<ul style="list-style-type: none"><li>Adds QEMU information, XenAPI information, EC2 API configuration, rootwrap configuration, updates to cloupipe information, dnsmasq information, expands network troubleshooting, and other various doc improvements.</li></ul>
May 3, 2012	<ul style="list-style-type: none"><li>Begin trunk designation.</li></ul>
May 2, 2012	<ul style="list-style-type: none"><li>Essex release.</li></ul>
May 1, 2012	<ul style="list-style-type: none"><li>Fixed PDF link.</li></ul>
Apr 25, 2012	<ul style="list-style-type: none"><li>Adds listing of all 467 configuration options in a new chapter.</li></ul>
Mar 20, 2012	<ul style="list-style-type: none"><li>Large reorganization, moving identity and image service administration as chapters in this book.</li></ul>
Nov 15, 2011	<ul style="list-style-type: none"><li>Added RSS feed and PDF link to header.</li></ul>
Nov 9, 2011	<ul style="list-style-type: none"><li>Updates for auth section regarding EC2 creds and drafted info about IP address association.</li><li>Compute conceptual info on projects and quotas and RBAC updated to describe deprecated auth vs identity service concepts.</li></ul>

# 1. Getting Started with OpenStack

## Table of Contents

Why Cloud? .....	1
What is OpenStack? .....	2
Components of OpenStack .....	3
Conceptual Architecture .....	3
Logical Architecture .....	4
Storage Concepts .....	10

OpenStack is a collection of open source technologies that provides massively scalable cloud computing software. OpenStack can be used by corporations, service providers, VARS, SMBs, researchers, and global data centers looking to deploy large-scale cloud deployments for private or public clouds.

## Why Cloud?

In data centers today, many computers suffer the same under-utilization in computing power and networking bandwidth. For example, projects may need a large amount of computing capacity to complete a computation, but no longer need the computing power after completing the computation. You want cloud computing when you want a service that's available on-demand with the flexibility to bring it up or down through automation or with little intervention. The phrase "cloud computing" is often represented with a diagram that contains a cloud-like shape indicating a layer where responsibility for service goes from user to provider. The cloud in these types of diagrams contains the services that afford computing power harnessed to get work done. Much like the electrical power we receive each day, cloud computing provides subscribers or users with access to a shared collection of computing resources: networks for transfer, servers for storage, and applications or services for completing tasks.

These are the compelling features of a cloud:

- On-demand self-service: Users can provision servers and networks with little human intervention.
- Network access: Any computing capabilities are available over the network. Many different devices are allowed access through standardized mechanisms.
- Resource pooling: Multiple users can access clouds that serve other consumers according to demand.
- Elasticity: Provisioning is rapid and scales out or in based on need.
- Metered or measured service: Just like utilities that are paid for by the hour, clouds should optimize resource use and control it for the level of service or type of servers such as storage or processing.

Cloud computing offers different service models depending on the capabilities a consumer may require.

- SaaS: Software as a Service. Provides the consumer the ability to use the software in a cloud environment, such as web-based email for example.
- PaaS: Platform as a Service. Provides the consumer the ability to deploy applications through a programming language or tools supported by the cloud platform provider. An example of platform as a service is an Eclipse/Java programming platform provided with no downloads required.
- IaaS: Infrastructure as a Service. Provides infrastructure such as computer instances, network connections, and storage so that people can run any software or operating system.

When you hear terms such as public cloud or private cloud, these refer to the deployment model for the cloud. A private cloud operates for a single organization, but can be managed on-premise or off-premise. A public cloud has an infrastructure that is available to the general public or a large industry group and is likely owned by a cloud services company. The NIST also defines community cloud as shared by several organizations supporting a specific community with shared concerns.

Clouds can also be described as hybrid. A hybrid cloud can be a deployment model, as a composition of both public and private clouds, or a hybrid model for cloud computing may involve both virtual and physical servers.

What have people done with cloud computing? Cloud computing can help with large-scale computing needs or can lead consolidation efforts by virtualizing servers to make more use of existing hardware and potentially release old hardware from service. People also use cloud computing for collaboration because of its high availability through networked computers. Productivity suites for word processing, number crunching, and email communications, and more are also available through cloud computing. Cloud computing also avails additional storage to the cloud user, avoiding the need for additional hard drives on each user's desktop and enabling access to huge data storage capacity online in the cloud.

For a more detailed discussion of cloud computing's essential characteristics and its models of service and deployment, see <http://www.nist.gov/itl/cloud/>, published by the US National Institute of Standards and Technology.

## What is OpenStack?

OpenStack is on a mission: to provide scalable, elastic cloud computing for both public and private clouds, large and small. At the heart of our mission is a pair of basic requirements: clouds must be simple to implement and massively scalable.

If you are new to OpenStack, you will undoubtedly have questions about installation, deployment, and usage. It can seem overwhelming at first. But don't fear, there are places to get information to guide you and to help resolve any issues you may run into during the on-ramp process. Because the project is so new and constantly changing, be aware of the revision time for all information. If you are reading a document that is a few months old and you feel that it isn't entirely accurate, then please let us know through the mailing list at <https://launchpad.net/~openstack> or by filing a bug at <https://bugs.launchpad.net/openstack-manuals/+filebug> so it can be updated or removed.

# Components of OpenStack

There are currently seven core components of OpenStack: Compute, Object Storage, Identity, Dashboard, Block Storage, Network and Image Service. Let's look at each in turn.

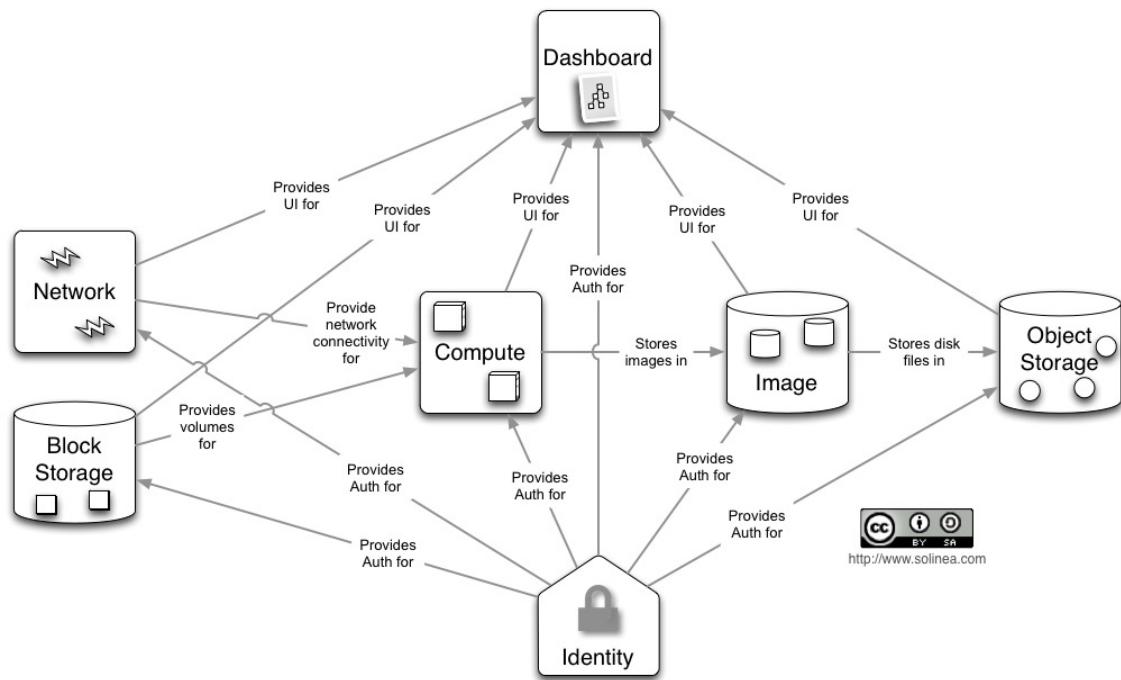
- Object Store (codenamed "[Swift](#)") provides object storage. It allows you to store or retrieve files (but not mount directories like a fileserver). Several companies provide commercial storage services based on Swift. These include KT, Rackspace (from which Swift originated) and Internap. Swift is also used internally at many large companies to store their data.
- Image (codenamed "[Glance](#)") provides a catalog and repository for virtual disk images. These disk images are mostly commonly used in OpenStack Compute. While this service is technically optional, any cloud of size will require it.
- Compute (codenamed "[Nova](#)") provides virtual servers upon demand. [Rackspace](#) and [HP](#) provide commercial compute services built on Nova and it is used internally at companies like Mercado Libre and NASA (where it originated).
- Dashboard (codenamed "[Horizon](#)") provides a modular web-based user interface for all the OpenStack services. With this web GUI, you can perform most operations on your cloud like launching an instance, assigning IP addresses and setting access controls.
- Identity (codenamed "[Keystone](#)") provides authentication and authorization for all the OpenStack services. It also provides a service catalog of services within a particular OpenStack cloud.
- Network (codenamed "[Quantum](#)") provides "network connectivity as a service" between interface devices managed by other OpenStack services (most likely Nova). The service works by allowing users to create their own networks and then attach interfaces to them. OpenStack Network has a pluggable architecture to support many popular networking vendors and technologies.
- Block Storage (codenamed "[Cinder](#)") provides persistent block storage to guest VMs.

In addition to these core projects, there are also a number of "incubation" projects that are being considered for future integration into the OpenStack release.

# Conceptual Architecture

The OpenStack project as a whole is designed to deliver a massively scalable cloud operating system. To achieve this, each of the constituent services are designed to work together to provide a complete Infrastructure as a Service (IaaS). This integration is facilitated through public application programming interfaces (APIs) that each service offers (and in turn can consume). While these APIs allow each of the services to use another service, it also allows an implementer to switch out any service as long as they maintain the API. These are (mostly) the same APIs that are available to end users of the cloud.

Conceptually, you can picture the relationships between the services as so:



- Dashboard ("Horizon") provides a web front end to the other OpenStack services
- Compute ("Nova") stores and retrieves virtual disks ("images") and associated metadata in Image ("Glance")
- Network ("Quantum") provides virtual networking for Compute.
- Block Storage ("Cinder") provides storage volumes for Compute.
- Image ("Glance") can store the actual virtual disk files in the Object Store("Swift")
- All the services authenticate with Identity ("Keystone")

This is a stylized and simplified view of the architecture, assuming that the implementer is using all of the services together in the most common configuration. It also only shows the "operator" side of the cloud – it does not picture how consumers of the cloud may actually use it. For example, many users will access object storage heavily (and directly).

## Logical Architecture

The following paragraphs give some details on the main modules in the OpenStack components.

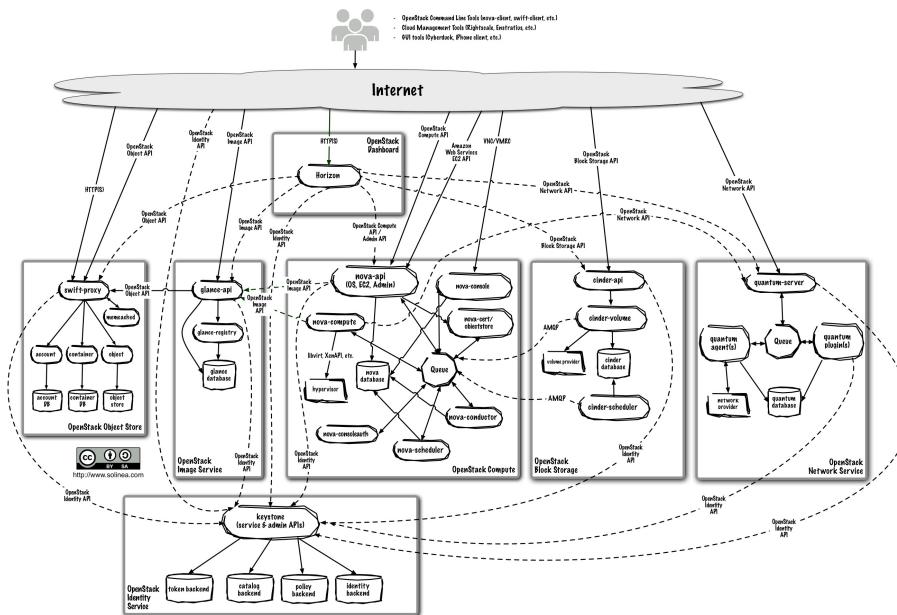
These details are not meant to be exhaustive; the objective is to describe the relevant aspects that administrators need to know to better understand how to design the deployment, and install and configure the whole platform.

Modules are organized according to the functional area they belong (i.e. the kind of functions they implement or deliver) and classified according to their type.

These are the types:

- daemon: runs as a daemon and, on Linux platforms, is usually installed as a service;
- script: a script run by an external module when some event happens (at the moment, it is used as a co-routine of dnsmasq for managing IP Addresses released to instances via DHCP protocol);
- client: a client for accessing the Python bindings for a service
- CLI: a Command Line Interpreter for submitting commands to OpenStack Compute for example

As you can imagine, the logical architecture is far more complicated than the conceptual architecture shown above. As with any service-oriented architecture, diagrams quickly become "messy" trying to illustrate all the possible combinations of service communications. The diagram below, illustrates the most common architecture of an OpenStack-based cloud. However, as OpenStack supports a wide variety of technologies, it does not represent the only architecture possible.



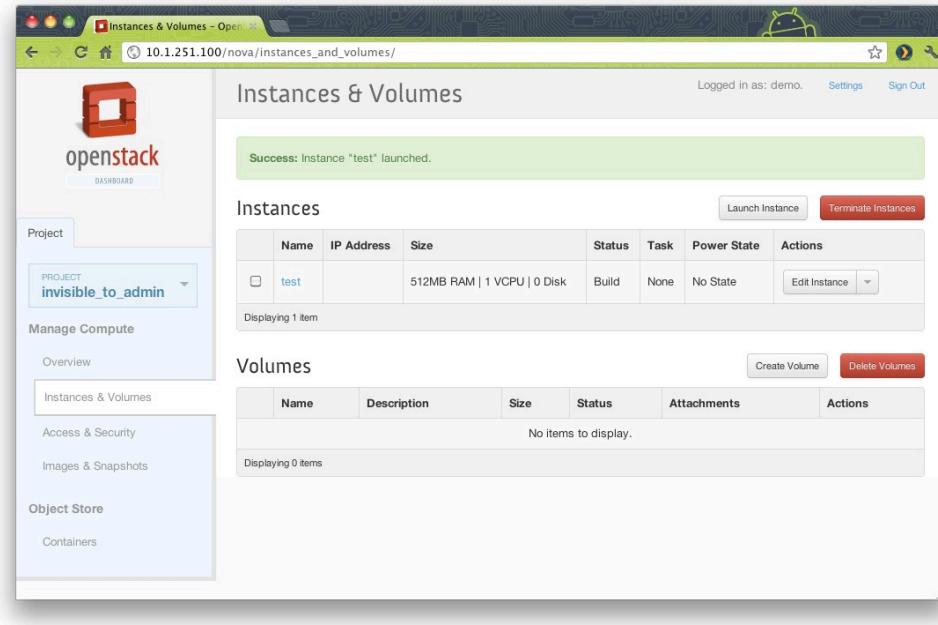
This picture is consistent with the conceptual architecture above in that:

- End users can interact through a common web interface (Horizon) or directly to each service through their API
- All services authenticate through a common source (facilitated through Keystone)
- Individual services interact with each other through their public APIs (except where privileged administrator commands are necessary)

In the sections below, we'll delve into the architecture for each of the services.

## Dashboard

Horizon is a modular [Django web application](#) that provides an end user and administrator interface to OpenStack services.



As with most web applications, the architecture is fairly simple:

- Horizon is usually deployed via [mod\\_wsgi](#) in Apache. The code itself is separated into a reusable python module with most of the logic (interactions with various OpenStack APIs) and presentation (to make it easily customizable for different sites).
- A database (configurable as to which one). As it relies mostly on the other services for data, it stores very little data of its own.

From a network architecture point of view, this service will need to be customer accessible as well as be able to talk to each service's public APIs. If you wish to use the administrator functionality (i.e. for other services), it will also need connectivity to their Admin API endpoints (which should be non-customer accessible).

## Compute

Nova is the most complicated and distributed component of OpenStack. A large number of processes cooperate to turn end user API requests into running virtual machines. Main modules are implemented in Python. The following lists are organized by functional areas:

### API

- `nova-api` accepts and responds to end user compute API calls. It supports OpenStack Compute API, Amazon's EC2 API and a special Admin API (for privileged users to perform administrative actions). It also initiates most of the orchestration activities (such as running an instance) as well as enforces some policy.
- `nova-api-metadata` accepts metadata requests from instances ([more details](#)). The `nova-api-metadata` service is generally only used when running in multi-host mode with `nova-network` installations.

### Computing core

- The `nova-compute` process is primarily a worker daemon that creates and terminates virtual machine instances via hypervisor's APIs (XenAPI for XenServer/XCP, libvirt for KVM or QEMU, VMwareAPI for VMware, etc.). The process by which it does so is fairly complex but the basics are simple: accept actions from the queue and then perform a series of system commands (like launching a KVM instance) to carry them out while updating state in the database.
- The `nova-schedule` process is conceptually the simplest piece of code in OpenStack Nova: take a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on).
- The `nova-conductor` module, introduced in the Grizzly release, works as a "mediator" between `nova-compute` and the database. It is aimed at eliminating all the direct accesses to the cloud database made by `nova-compute`. The `nova-conductor` module scales horizontally but it shouldn't be deployed on the same node(s) where `nova-compute` runs. You can [read more about the new service here](#).

## Networking for VMs

- The `nova-network` worker daemon is very similar to `nova-compute`. It accepts networking tasks from the queue and then performs tasks to manipulate the network (such as setting up bridging interfaces or changing iptables rules). This functionality is being migrated to OpenStack Networking, a separate OpenStack service.
- `nova-dhcpbridge` ([script](#)) This script tracks IP address leases and records them in the database using dnsmasq's dhcp-script facility. This functionality is also migrated to OpenStack Networking; a different script is provided when using OpenStack Networking (code-named Quantum).

## Console Interface

- The `nova-consoleauth` daemon authorizes user's tokens that console proxies provide (see `nova-novncproxy` and `nova-xvpnvncproxy`). This service must be running in order for console proxies to work. Many proxies of either type can be run against a single `nova-consoleauth` service in a cluster configuration. [Read more details](#).
- The `nova-novncproxy` ([daemon](#)) provides a proxy for accessing running instances through a VNC connection. It supports browser-based novnc clients.
- The deprecated `nova-console` daemon is no longer used with Grizzly, and the `nova-xvpnvncproxy` is used instead.
- The `nova-xvpnvncproxy` daemon is a proxy for accessing running instances through a VNC connection. It supports a Java client specifically designed for OpenStack.
- The `nova-cert` daemon manages x509 certificates.

## Image Management (EC2 scenario)

- The `nova-objectstore` daemon provides an S3 interface for registering images onto the image management service (see `glance`) It is mainly used for installations that need to support euca2ools. The euca2ools tools talk to `nova-objectore` in "S3 language" and `nova-objectstore` translates S3 requests into `glance` requests

- The `euca2ools` client is not an OpenStack module but it can be supported by OpenStack. It's a set of command line interpreter commands for managing cloud resources. Provided that `nova-api` is configured to support EC2 interface, `euca2ools` can be used to issue cloud management commands. For more information on `euca2ools`, see <http://www.eucalyptus.com/eucalyptus-cloud/documentation/2.0>.

### Command Line Interpreter/Interfaces

- The `nova` client enables you to submit either tenant administrator's commands or cloud user's commands.
- The `nova-manage` client submits cloud administrator commands.
- The queue provides a central hub for passing messages between daemons. This is usually implemented with [RabbitMQ](#) today, but could be any AMPQ message queue (such as [Apache Qpid](#)), or [Zero MQ](#).
- The SQL database stores most of the build-time and run-time state for a cloud infrastructure. This includes the instance types that are available for use, instances in use, networks available and projects. Theoretically, OpenStack Nova can support any database supported by SQL-Alchemy but the only databases currently being widely used are `sqlite3` (only appropriate for test and development work), MySQL and PostgreSQL.

Nova interacts with many other OpenStack services: Keystone for authentication, Glance for images and Horizon for web interface. The Glance interactions are central. The API process can upload and query Glance while `nova-compute` will download images for use in launching images.

## Object Store

The swift architecture is very distributed to prevent any single point of failure as well as to scale horizontally. It includes the following components:

- Proxy server (`swift-proxy-server`) accepts incoming requests via the OpenStack Object API or just raw HTTP. It accepts files to upload, modifications to metadata or container creation. In addition, it will also serve files or container listing to web browsers. The proxy server may utilize an optional cache (usually deployed with memcache) to improve performance.
- Account servers manage accounts defined with the object storage service.
- Container servers manage a mapping of containers (i.e folders) within the object store service.
- Object servers manage actual objects (i.e. files) on the storage nodes.
- There are also a number of periodic processes which run to perform housekeeping tasks on the large data store. The most important of these is the replication services, which ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters and reapers.

Authentication is handled through configurable WSGI middleware (which will usually be Keystone).

## Image Store

Glance has four main parts to it:

- `glance-api` accepts Image API calls for image discovery, image retrieval and image storage.
- `glance-registry` stores, processes and retrieves metadata about images (size, type, etc.).
- A database to store the image metadata. Like Nova, you can choose your database depending on your preference (but most people use MySQL or SQLite).
- A storage repository for the actual image files. In the diagram above, Swift is shown as the image repository, but this is configurable. In addition to Swift, Glance supports normal filesystems, RADOS block devices, Amazon S3 and HTTP. Be aware that some of these choices are limited to read-only usage.

There are also a number of periodic process which run on Glance to support caching. The most important of these is the replication services, which ensures consistency and availability through the cluster. Other periodic processes include auditors, updaters and reapers.

As you can see from the diagram in the Conceptual Architecture section, Glance serves a central role to the overall IaaS picture. It accepts API requests for images (or image metadata) from end users or Nova components and can store its disk files in the object storage service, Swift.

## Identity

Keystone provides a single point of integration for OpenStack policy, catalog, token and authentication.

- `keystone` handles API requests as well as providing configurable catalog, policy, token and identity services.
- Each Keystone function has a pluggable backend which allows different ways to use the particular service. Most support standard backends like LDAP or SQL, as well as Key Value Stores (KVS).

Most people will use this as a point of customization for their current authentication services.

## Network

OpenStack Networking provides "network connectivity as a service" between interface devices managed by other OpenStack services (most likely Compute). The service works by allowing users to create their own networks and then attach interfaces to them. Like many of the OpenStack services, OpenStack Networking is highly configurable due to its plug-in architecture. These plug-ins accommodate different networking equipment and software. As such, the architecture and deployment can vary dramatically. In the above architecture, a simple Linux networking plug-in is shown.

- `quantum-server` accepts API requests and then routes them to the appropriate OpenStack Networking plugin for action.
- OpenStack Networking plugins and agents perform the actual actions such as plugging and unplugging ports, creating networks or subnets and IP addressing. These plugins and agents differ depending on the vendor and technologies used in the particular cloud. OpenStack Networking ships with plugins and agents for: Cisco virtual and physical switches, Nicira NVP product, NEC OpenFlow products, Open vSwitch, Linux bridging and the Ryu Network Operating System.

The common agents are L3 (layer 3), DHCP (dynamic host IP addressing) and the specific plug-in agent.

- Most OpenStack Networking installations also make use of a messaging queue to route information between the `quantum-server` and various agents as well as a database to store networking state for particular plugins.

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

## Block Storage

The OpenStack Block Storage API allows for manipulation of volumes, volume types (similar to compute flavors) and volume snapshots.

- `cinder-api` accepts API requests and routes them to `cinder-volume` for action.
- `cinder-volume` acts upon the requests by reading or writing to the Cinder database to maintain state, interacting with other processes (like `cinder-scheduler`) through a message queue and directly upon block storage providing hardware or software. It can interact with a variety of storage providers through a driver architecture. Currently, there are drivers for IBM, SolidFire, NetApp, Nexenta, Zadara, GlusterFS, linux iSCSI and other storage providers.
- Much like `nova-scheduler`, the `cinder-scheduler` daemon picks the optimal block storage provider node to create the volume on.
- OpenStack Block Storage deployments will also make use of a messaging queue to route information between the `cinder` processes as well as a database to store volume state.

Like OpenStack Network, OpenStack Block Storage will mainly interact with OpenStack Compute, providing volumes for its instances.

## Storage Concepts

Storage is found in many parts of the OpenStack stack, and the differing types can cause confusion to even experienced cloud engineers. Here's a simple chart to kick-start your understanding:

**Table 1.1. Types of Storage**

On-instance / ephemeral	Volumes block storage (Cinder)	Object Storage (Swift)
Used for running Operating System and scratch space	Used for adding additional persistent storage to a virtual machine (VM)	Used for storing virtual machine images and data

On-instance / ephemeral	Volumes block storage (Cinder)	Object Storage (Swift)
Persists until VM is terminated	Persists until deleted	Persists until deleted
Access associated with a VM	Access associated with a VM	Available from anywhere
Implemented as a filesystem underlying OpenStack Compute	Mounted via OpenStack Block-Storage controlled protocol (for example, iSCSI)	REST API
Administrator configures size setting, based on flavors	Sizings based on need	Easily scalable for future growth
Example: 10GB first disk, 30GB/core second disk	Example: 1TB "extra hard drive"	Example: 10s of TBs of dataset storage

Other points of note include:

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

## 2. Introduction to OpenStack Compute

### Table of Contents

Hypervisors .....	12
Users and Tenants (Projects) .....	12
Images and Instances .....	13
System Architecture .....	16
Block Storage and OpenStack Compute .....	16

OpenStack Compute gives you a tool to orchestrate a cloud, including running instances, managing networks, and controlling access to the cloud through users and projects. The underlying open source project's name is Nova, and it provides the software that can control an Infrastructure as a Service (IaaS) cloud computing platform. It is similar in scope to Amazon EC2 and Rackspace Cloud Servers. OpenStack Compute does not include any virtualization software; rather it defines drivers that interact with underlying virtualization mechanisms that run on your host operating system, and exposes functionality over a web-based API.

### Hypervisors

OpenStack Compute requires a hypervisor and Compute controls the hypervisors through an API server. The process for selecting a hypervisor usually means prioritizing and making decisions based on budget and resource constraints as well as the inevitable list of supported features and required technical specifications. The majority of development is done with the KVM and Xen-based hypervisors. Refer to <http://wiki.openstack.org/HypervisorSupportMatrix> for a detailed list of features and support across the hypervisors.

With OpenStack Compute, you can orchestrate clouds using multiple hypervisors in different zones. The types of virtualization standards that may be used with Compute include:

- [KVM](#) - Kernel-based Virtual Machine
- [LXC](#) - Linux Containers (through libvirt)
- [QEMU](#) - Quick EMULATOR
- [UML](#) - User Mode Linux
- [VMWare vSphere](#) 4.1 update 1 and newer
- [Xen](#) - Xen, Citrix XenServer and Xen Cloud Platform (XCP)
- [Bare Metal](#) - Provisions physical hardware via pluggable sub-drivers.

### Users and Tenants (Projects)

The OpenStack Compute system is designed to be used by many different cloud computing consumers or customers, basically tenants on a shared system, using role-based access

assignments. Roles control the actions that a user is allowed to perform. In the default configuration, most actions do not require a particular role, but this is configurable by the system administrator editing the appropriate `policy.json` file that maintains the rules. For example, a rule can be defined so that a user cannot allocate a public IP without the admin role. A user's access to particular images is limited by tenant, but the username and password are assigned per user. Key pairs granting access to an instance are enabled per user, but quotas to control resource consumption across available hardware resources are per tenant.



### Note

Earlier versions of OpenStack used the term "project" instead of "tenant". Because of this legacy terminology, some command-line tools use `--project_id` when a tenant ID is expected.

While the original EC2 API supports users, OpenStack Compute adds the concept of tenants. Tenants are isolated resource containers forming the principal organizational structure within the Compute service. They consist of a separate VLAN, volumes, instances, images, keys, and users. A user can specify which tenant he or she wishes to be known as by appending `:project_id` to his or her 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, quota controls are available to limit the:

- Number of volumes which may be created
- Total size of all volumes within a project as measured in GB
- Number of instances which may be launched
- Number of processor cores which may be allocated
- Floating IP addresses (assigned to any instance when it launches so the instance has the same publicly accessible IP addresses)
- Fixed IP addresses (assigned to the same instance each time it boots, publicly or privately accessible, typically private for management purposes)

## Images and Instances

This introduction provides a high level overview of what images and instances are and description of the life-cycle of a typical virtual system within the cloud. There are many ways to configure the details of an OpenStack cloud and many ways to implement a virtual system within that cloud. These configuration details as well as the specific command line utilities and API calls to preform the actions described are presented in the [Image Management](#) and [Volume Management](#) chapters.

Images are disk images which are templates for virtual machine file systems. The image service, Glance, is responsible for the storage and management of images within OpenStack.

Instances are the individual virtual machines running on physical compute nodes. The compute service, Nova, manages instances. Any number of instances maybe started

from the same image. Each instance is run from a copy of the base image so runtime changes made by an instance do not change the image it is based on. Snapshots of running instances may be taken which create a new image based on the current disk state of a particular instance.

When starting an instance a set of virtual resources known as a flavor must be selected. Flavors define how many virtual CPUs an instance has and the amount of RAM and size of its ephemeral disks. OpenStack provides a number of predefined flavors which cloud administrators may edit or add to. Users must select from the set of available flavors defined on their cloud.

Additional resources such as persistent volume storage and public IP address may be added to and removed from running instances. The examples below show the cinder-volume service which provide persistent block storage as opposed to the ephemeral storage provided by the instance flavor.

Here is an example of the life cycle of a typical virtual system within an OpenStack cloud to illustrate these concepts.

## Initial State

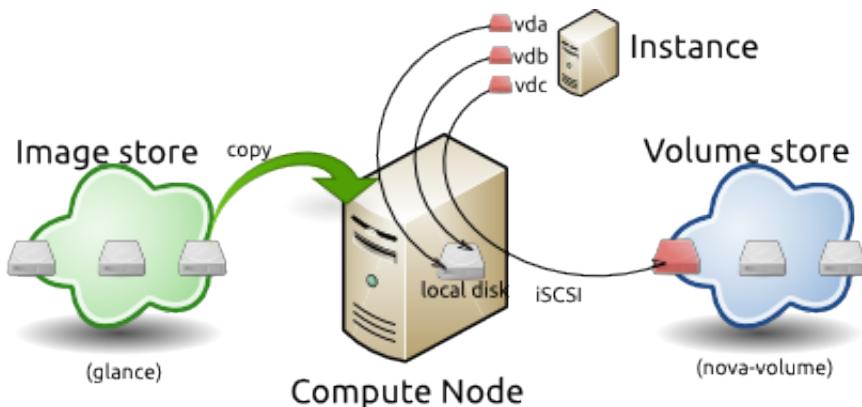
The following diagram shows the system state prior to launching an instance. The image store fronted by the image service, Glance, has some number of predefined images. In the cloud there is an available compute node with available vCPU, memory and local disk resources. Plus there are a number of predefined volumes in the cinder-volume service.

**Figure 2.1. Base image state with no running instances**



## Launching an instance

To launch an instance the user selects an image, a flavor and optionally other attributes. In this case the selected flavor provides a root volume (as all flavors do) labeled vda in the diagram and additional ephemeral storage labeled vdb in the diagram. The user has also opted to map a volume from the cinder-volume store to the third virtual disk, vdc, on this instance.

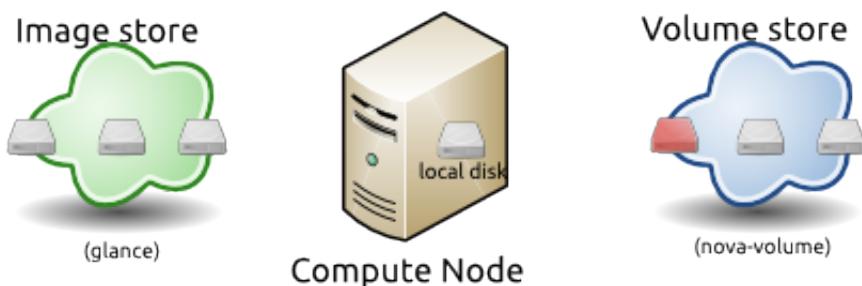
**Figure 2.2. Instance creation from image and run time state**

The OpenStack system copies the base image from the image store to local disk which is used as the first disk of the instance (vda), having small images will result in faster start up of your instances as less data needs to be copied across the network. The system also creates a new empty disk image to present as the second disk (vdb). Be aware that the second disk is an empty disk with an ephemeral life as it is destroyed when you delete the instance. The compute node attaches to the requested cinder-volume using iSCSI and maps this to the third disk (vdc) as requested. The vCPU and memory resources are provisioned and the instance is booted from the first drive. The instance runs and changes data on the disks indicated in red in the diagram.

There are many possible variations in the details of the scenario, particularly in terms of what the backing storage is and the network protocols used to attach and move storage. One variant worth mentioning here is that the ephemeral storage used for volumes vda and vdb in this example may be backed by network storage rather than local disk. The details are left for later chapters.

## End State

Once the instance has served its purpose and is deleted all state is reclaimed, except the persistent volume. The ephemeral storage is purged. Memory and vCPU resources are released. And of course the image has remained unchanged through out.

**Figure 2.3. End state of image and volume after instance exits**

## System Architecture

OpenStack Compute consists of several main components. A "cloud controller" contains many of these components, and it represents the global state and interacts with all other components. An API Server acts as the web services front end for the cloud controller. The compute controller provides compute server resources and typically contains the compute service, The Object Store component optionally provides storage services. An auth manager provides authentication and authorization services when used with the Compute system, or you can use the Identity Service (keystone) as a separate authentication service. A volume controller provides fast and permanent block-level storage for the compute servers. A network controller provides virtual networks to enable compute servers to interact with each other and with the public network. A scheduler selects the most suitable compute controller to host an instance.

OpenStack Compute is built on a shared-nothing, messaging-based architecture. You can run all of the major components on multiple servers including a compute controller, volume controller, network controller, and object store (or image service). A cloud controller communicates with the internal object store via HTTP (Hyper Text Transfer Protocol), but it communicates with a scheduler, network controller, and volume controller via AMQP (Advanced Message Queue Protocol). To avoid blocking each component while waiting for a response, OpenStack Compute uses asynchronous calls, with a call-back that gets triggered when a response is received.

To achieve the shared-nothing property with multiple copies of the same component, OpenStack Compute keeps all the cloud system state in a database.

## Block Storage and OpenStack Compute

OpenStack provides two classes of block storage, "ephemeral" storage and persistent "volumes". Ephemeral storage exists only for the life of an instance, it will persist across reboots of the guest operating system but when the instance is deleted so is the associated storage. All instances have some ephemeral storage. Volumes are persistent virtualized block devices independent of any particular instance. Volumes may be attached to a single instance at a time, but may be detached or reattached to a different instance while retaining all data, much like a USB drive.

### Ephemeral Storage

Ephemeral storage is associated with a single unique instance. Its size is defined by the flavor of the instance.

Data on ephemeral storage ceases to exist when the instance it is associated with is terminated. Rebooting the VM or restarting the host server, however, will not destroy ephemeral data. In the typical use case an instance's root filesystem is stored on ephemeral storage. This is often an unpleasant surprise for people unfamiliar with the cloud model of computing.

In addition to the ephemeral root volume all flavors except the smallest, m1.tiny, provide an additional ephemeral block device varying from 20G for the m1.small through 160G for the m1.xlarge by default - these sizes are configurable. This is presented as a raw block

---

device with no partition table or filesystem. Cloud aware operating system images may discover, format, and mount this device. For example the cloud-init package included in Ubuntu's stock cloud images will format this space as an ext3 filesystem and mount it on /mnt. It is important to note this a feature of the guest operating system. OpenStack only provisions the raw storage.

## Volume Storage

Volume storage is independent of any particular instance and is persistent. Volumes are user created and within quota and availability limits may be of any arbitrary size.

When first created volumes are raw block devices with no partition table and no filesystem. They must be attached to an instance to be partitioned and/or formatted. Once this is done they may be used much like an external disk drive. Volumes may attached to only one instance at a time, but may be detached and reattached to either the same or different instances.

It is possible to configure a volume so that it is bootable and provides a persistent virtual instance similar to traditional non-cloud based virtualization systems. In this use case the resulting instance may still have ephemeral storage depending on the flavor selected, but the root filesystem (and possibly others) will be on the persistent volume and thus state will be maintained even if the instance is shutdown. Details of this configuration are discussed in the [Boot From Volume](#) section of this manual.

Volumes do not provide concurrent access from multiple instances. For that you need either a traditional network filesystem like NFS or CIFS or a cluster filesystem such as GlusterFS. These may be built within an OpenStack cluster or provisioned outside of it, but are not features provided by the OpenStack software.

# 3. Installing OpenStack Compute

## Table of Contents

Compute and Image System Requirements .....	18
Installing on openSUSE or SUSE Linux Enterprise Server .....	19
Installing OpenStack Compute on Debian .....	20
Installing on Citrix XenServer .....	21

The OpenStack system has several key projects that are separate installations but can work together depending on your cloud needs: OpenStack Compute, OpenStack Block Storage, OpenStack Object Storage, and the OpenStack Image Service. You can install any of these projects separately and then configure them either as standalone or connected entities. The installation process is documented in the OpenStack Install Guide, for either Ubuntu or RHEL/CentOS/Fedora. You can choose your guide at [docs.openstack.org/install](http://docs.openstack.org/install).

## Compute and Image System Requirements

**Hardware:** OpenStack components are intended to run on standard hardware.

Recommended hardware configurations for a minimum production deployment are as follows for the cloud controller nodes and compute nodes for Compute and the Image Service, and object, account, container, and proxy servers for Object Storage.

**Table 3.1. Hardware Recommendations**

Server	Recommended Hardware	Notes
Cloud Controller node (runs network, volume, API, scheduler and image services)	Processor: 64-bit x86  Memory: 12 GB RAM  Disk space: 30 GB (SATA, SAS or SSD)  Volume storage: two disks with 2 TB (SATA) for volumes attached to the compute nodes  Network: one 1 Gbps Network Interface Card (NIC)	Two NICs are recommended but not required. A quad core server with 12 GB RAM would be more than sufficient for a cloud controller node.
Compute nodes (runs virtual instances)	Processor: 64-bit x86  Memory: 32 GB RAM  Disk space: 30 GB (SATA)  Network: two 1 Gbps NICs	With 2 GB RAM you can run one m1.small instance on a node or three m1.tiny instances without memory swapping, so 2 GB RAM would be a minimum for a test-environment compute node. As an example, Rackspace Cloud Builders use 96 GB RAM for compute nodes in OpenStack deployments.  Specifically for virtualization on certain hypervisors on the node or nodes running nova-compute, you need a x86 machine with an AMD processor with SVM extensions (also called AMD-V) or an Intel processor with VT (virtualization technology) extensions.  For XenServer and XCP refer to the <a href="#">XenServer installation guide</a> and the <a href="#">XenServer hardware compatibility list</a> .  For LXC, the VT extensions are not required.



### Note

While certain parts of OpenStack are known to work on various operating systems, currently the only feature-complete, production-supported host environment is 64-bit Linux.

**Operating System:** OpenStack currently has packages for the following distributions: CentOS, Debian, Fedora, RHEL, openSUSE, SLES, and Ubuntu. These packages are maintained by community members, refer to <http://wiki.openstack.org/Packaging> for additional links.



### Note

The Grizzly version is available on the most recent LTS (Long Term Support) version which is 12.04 (Precise Pangolin), via the Ubuntu Cloud Archive. At this time, there are not packages available for 12.10. It is also available on the current Ubuntu development series, which is 13.04 (Raring Ringtail).

The Grizzly release of OpenStack Compute requires Fedora 16 or later.

**Database:** For OpenStack Compute, you need access to either a PostgreSQL or MySQL database, or you can install it as part of the OpenStack Compute installation process.

**Permissions:** You can install OpenStack services either as root or as a user with sudo permissions if you configure the sudoers file to enable all the permissions.

**Network Time Protocol:** You must install a time synchronization program such as NTP. For Compute, time synchronization avoids problems when scheduling VM launches on compute nodes. For Object Storage, time synchronization ensure the object replications are accurately updating objects when needed so that the freshest content is served.

## Installing on openSUSE or SUSE Linux Enterprise Server

The [openSUSE Build Service](#) is used for packaging and publishing of different OpenStack versions for openSUSE and SUSE Linux Enterprise releases. It thus allows to install newer OpenStack releases.

Packages are created in subprojects of [Cloud:OpenStack](#) named after the OpenStack release, e.g. [Cloud:OpenStack:Grizzly](#).

[B1 Systems GmbH](#) provides packages for openSUSE 12.2 and SUSE Linux Enterprise Server 11 SP2 on the [openSUSE Open Build Server](#).

These packages are currently build for active openSUSE releases and SUSE Linux Enterprise 11 and can be downloaded from [download.opensuse.org](http://download.opensuse.org).

## SUSE Linux Enterprise Server

Declare the repository to libzypp with **zypper ar**.

```
# zypper ar -f http://download.opensuse.org/repositories/Cloud:/OpenStack:/  
Grizzly/SLE_11_SP2/Cloud:OpenStack:Grizzly.repo  
  
Adding repository 'OpenStack Grizzly (Stable branch) (SLE_11_SP2)' .....  
.....[done]  
Repository 'OpenStack Grizzly (Stable branch) (SLE_11_SP2)' successfully added  
Enabled: Yes  
Autorefresh: Yes  
GPG check: Yes  
URI: http://download.opensuse.org/repositories/Cloud:/OpenStack:/Grizzly/  
SLE_11_SP2/
```

You can list all available packages for OpenStack with **zypper se openstack**. You can install packages with **zypper in PACKAGE**.



## Warning

You have to apply the latest available updates for SLES11 SP2. Without doing that it's not possible to run OpenStack on SLES11 SP2. For evaluation purposes you can request a [free 60 day evaluation for SLES11 SP2](#) to gain updates.

To verify that you use the correct Python interpreter simply check the version. You should use at least Python 2.6.8.

```
# python --version  
Python 2.6.8
```

## openSUSE

Declare the repository to libzypp with **zypper ar**.

```
# zypper ar -f http://download.opensuse.org/repositories/Cloud:/OpenStack:/  
Grizzly/openSUSE_12.3/Cloud:OpenStack:Grizzly.repo  
Adding repository 'OpenStack Grizzly (Stable branch) (openSUSE_12.3)' .....  
.....[done]  
Repository 'OpenStack Grizzly (Stable branch) (openSUSE_12.3)' successfully  
added  
Enabled: Yes  
Autorefresh: Yes  
GPG check: Yes  
URI: http://download.opensuse.org/repositories/Cloud:/OpenStack:/Grizzly/  
openSUSE_12.3/
```

You can list all available packages for OpenStack with **zypper se openstack**. You can install packages with **zypper in PACKAGE**.

## Installing OpenStack Compute on Debian

Starting with Debian 7.0 "Wheezy", the OpenStack packages are provided as part of the distribution. Some non-official packages for Grizzly on Debian are available here:

```
deb http://archive.gplhost.com/debian grizzly main
deb http://archive.gplhost.com/debian grizzly-backports main
```

For the management or controller node install the following packages: (via `apt-get install`)

- `nova-api`
- `nova-scheduler`
- `nova-conductor`
- `glance`
- `keystone`
- `mysql-server`
- `rabbitmq`
- `memcached`
- `openstack-dashboard`

For the compute node(s) install the following packages:

- `nova-compute`
- `nova-network`
- `nova-api`



### Note

Because this manual takes active advantage of the "sudo" command, it would be easier for you to add to it your Debian system, by doing:

```
# usermod -a -G sudo "myuser"
```

then re-login. Otherwise you will have to replace every "sudo" call by executing from root account.

## Installing on Citrix XenServer

When using OpenStack Compute with Citrix XenServer or XCP hypervisor, OpenStack Compute should be installed in a virtual machine running on your hypervisor, rather than installed directly on the hypervisor, as you would do when using the Libvirt driver. For more information see: [XenAPI Install](#).

Given how you should deploy OpenStack with XenServer, the first step when setting up the compute nodes in your OpenStack cloud is to install XenServer and install the required XenServer plugins. You can install XCP by installing Debian or Ubuntu, but generally rather than installing the operating system of your choice on your compute nodes, you should first install XenServer. For more information see: [XenAPI Deployment Architecture](#).

---

Once you have installed XenServer and the XenAPI plugins on all your compute nodes, you next need to create a virtual machine on each of those compute nodes. This must be a Linux virtual machine running in para-virtualized mode. It is inside each of these VMs that you will run the OpenStack components. You can follow the previous distribution specific instructions to get the OpenStack code running in your Virtual Machine. Once installed, you will need to configure OpenStack Compute to talk to your XenServer or XCP installation. For more information see: [Introduction to Xen](#).

# 4. Configuring OpenStack Compute

## Table of Contents

Post-Installation Configuration for OpenStack Compute .....	23
General Compute Configuration Overview .....	30
Example <code>nova.conf</code> Configuration Files .....	31
Configuring Logging .....	34
Configuring Hypervisors .....	35
Configuring Authentication and Authorization .....	38
Configuring Compute to use IPv6 Addresses .....	40
Configuring Image Service and Storage for Compute .....	41
Configuring Migrations .....	41
Configuring Resize .....	47
Installing MooseFS as shared storage for the instances directory .....	47
Configuring Database Connections .....	52
Configuring the Oslo RPC Messaging System .....	53
Configuring the Compute API .....	56
Configuring the EC2 API .....	58
Configuring Quotas .....	58

The OpenStack system has several key projects that are separate installations but can work together depending on your cloud needs: OpenStack Compute, OpenStack Object Storage, and OpenStack Image Store. There are basic configuration decisions to make, and the [OpenStack Install Guide](#) covers a basic walkthrough.

## Post-Installation Configuration for OpenStack Compute

Configuring your Compute installation involves many configuration files - the `nova.conf` file, the `api-paste.ini` file, and related Image and Identity management configuration files. This section contains the basics for a simple multi-node installation, but Compute can be configured many ways. You can find networking options and hypervisor options described in separate chapters.

### Setting Configuration Options in the `nova.conf` File

The configuration file `nova.conf` is installed in `/etc/nova` by default. A default set of options are already configured in `nova.conf` when you install manually.

Starting with the default file, you must define the following required items in `/etc/nova/nova.conf`. The options are described below. You can place comments in the `nova.conf` file by entering a new line with a # sign at the beginning of the line. To see a listing of all possible configuration options, refer to the [Compute Options Reference](#).

Here is a simple example `nova.conf` file for a small private cloud, with all the cloud controller services, database server, and messaging server on the same server. In this case, `CONTROLLER_IP` represents the IP address of a central server, `BRIDGE_INTERFACE`

represents the bridge such as br100, the NETWORK\_INTERFACE represents an interface to your VLAN setup, and passwords are represented as DB\_PASSWORD\_COMPUTE for your Compute (nova) database password, and RABBIT PASSWORD represents the password to your rabbit installation.

```
[DEFAULT]

# LOGS/STATE
verbose=True
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
rootwrap_config=/etc/nova/rootwrap.conf

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
# configured in cinder.conf

# DATABASE
sql_connection=mysql://nova:yourpassword@192.168.206.130/nova

# COMPUTE
libvirt_type=qemu
compute_driver=libvirt.LibvirtDriver
instance_name_template=instance-%08x
api_paste_config=/etc/nova/api-paste.ini

# COMPUTE/APIS: if you have separate configs for separate services
# this flag is required for both nova-api and nova-compute
allow_resize_to_same_host=True

# APIS
osapi_compute_extension=nova.api.openstack.compute.contrib.standard_extensions
ec2_dmz_host=192.168.206.130
s3_host=192.168.206.130

# RABBITMQ
rabbit_host=192.168.206.130

# GLANCE
image_service=nova.image.glance.GlanceImageService
glance_api_servers=192.168.206.130:9292

# NETWORK
network_manager=nova.network.manager.FlatDHCPManager
force_dhcp_release=True
dhcpbridge_flagfile=/etc/nova/nova.conf
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
# Change my_ip to match each host
my_ip=192.168.206.130
public_interface=eth0
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=''

# NOVNC CONSOLE
```

```
novncproxy_base_url=http://192.168.206.130:6080/vnc_auto.html
# Change vncserver_proxyclient_address and vncserver_listen to match each
# compute host
vncserver_proxyclient_address=192.168.206.130
vncserver_listen=192.168.206.130

# AUTHENTICATION
auth_strategy=keystone
[keystone_authtoken]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = nova
signing dirname = /tmp/keystone-signing-nova
```



## Note

If your OpenStack deployment uses Qpid as the message queue instead of RabbitMQ (e.g., on Fedora, CentOS, RHEL), you would see `qpidd_hostname` instead of `rabbit_host` in the `nova.conf` file.

Create a `nova` group, so you can set permissions on the configuration file:

```
$ sudo addgroup nova
```

The `nova.conf` file should have its owner set to `root:nova`, and mode set to `0640`, since the file could contain your MySQL server's username and password. You also want to ensure that the `nova` user belongs to the `nova` group.

```
$ sudo usermod -g nova nova
$ chown -R username:nova /etc/nova
$ chmod 640 /etc/nova/nova.conf
```

# Setting Up OpenStack Compute Environment on the Compute Node

These are the commands you run to ensure the database schema is current:

```
$ nova-manage db sync
```

## Creating Credentials

The credentials you will use to launch instances, bundle images, and all the other assorted API functions can be sourced in a single file, such as creating one called `/creds/openrc`.

Here's an example `openrc` file you can download from the Dashboard in Settings > Project Settings > Download RC File.

```
#!/bin/bash
# *NOTE*: Using the 2.0 *auth api* does not mean that compute api is 2.0. We
# will use the 1.1 *compute api*
export OS_AUTH_URL=http://50.56.12.206:5000/v2.0
export OS_TENANT_ID=27755fd279ce43f9b17ad2d65d45b75c
export OS_USERNAME=vish
export OS_PASSWORD=$OS_PASSWORD_INPUT
export OS_AUTH_USER=norm
export OS_AUTH_KEY=$OS_PASSWORD_INPUT
export OS_AUTH_TENANT=27755fd279ce43f9b17ad2d65d45b75c
export OS_AUTH_STRATEGY=keystone
```

You also may want to enable EC2 access for the euca2ools. Here is an example `ec2rc` file for enabling EC2 access with the required credentials.

```
export NOVA_KEY_DIR=/root/creds/
export EC2_ACCESS_KEY="EC2KEY:USER"
export EC2_SECRET_KEY="SECRET_KEY"
export EC2_URL="http://$NOVA-API-IP:8773/services/Cloud"
export S3_URL="http://$NOVA-API-IP:3333"
export EC2_USER_ID=42 # nova does not use user id, but bundling requires it
export EC2_PRIVATE_KEY=${NOVA_KEY_DIR}/pk.pem
export EC2_CERT=${NOVA_KEY_DIR}/cert.pem
export NOVA_CERT=${NOVA_KEY_DIR}/cacert.pem
export EUCALYPTUS_CERT=${NOVA_CERT} # euca-bundle-image seems to require this
set
alias ec2-bundle-image="ec2-bundle-image --cert ${EC2_CERT} --privatekey
${EC2_PRIVATE_KEY} --user 42 --ec2cert ${NOVA_CERT}"
alias ec2-upload-bundle="ec2-upload-bundle -a ${EC2_ACCESS_KEY} -s
${EC2_SECRET_KEY} --url ${S3_URL} --ec2cert ${NOVA_CERT}"
```

Lastly, here is an example `openrc` file that works with nova client and ec2 tools.

```
export OS_PASSWORD=${ADMIN_PASSWORD:-secrete}
export OS_AUTH_URL=${OS_AUTH_URL:-http://$SERVICE_HOST:5000/v2.0}
export NOVA_VERSION=${NOVA_VERSION:-1.1}
export OS_REGION_NAME=${OS_REGION_NAME:-RegionOne}
export EC2_URL=${EC2_URL:-http://$SERVICE_HOST:8773/services/Cloud}
export EC2_ACCESS_KEY=${DEMO_ACCESS}
export EC2_SECRET_KEY=${DEMO_SECRET}
export S3_URL=http://$SERVICE_HOST:3333
export EC2_USER_ID=42 # nova does not use user id, but bundling requires it
export EC2_PRIVATE_KEY=${NOVA_KEY_DIR}/pk.pem
export EC2_CERT=${NOVA_KEY_DIR}/cert.pem
export NOVA_CERT=${NOVA_KEY_DIR}/cacert.pem
export EUCALYPTUS_CERT=${NOVA_CERT} # euca-bundle-image seems to require this
set
```

Next, add these credentials to your environment prior to running any nova client commands or nova commands.

```
$ cat /root/creds/openrc >> ~/.bashrc
source ~/.bashrc
```

## Creating Certificates

You can create certificates contained within pem files using these nova client commands, ensuring you have set up your environment variables for the nova client:

```
# nova x509-get-root-cert
# nova x509-create-cert
```

## Creating networks

You need to populate the database with the network configuration information that Compute obtains from the `nova.conf` file. You can find out more about the `nova network-create` command with `nova help network-create`.

Here is an example of what this looks like with real values entered. This example would be appropriate for FlatDHCP mode, for VLAN Manager mode you would also need to specify a VLAN.

```
$ nova network-create novanet --fixed-range-v4 192.168.0.0/24
```

For this example, the number of IPs is /24 since that falls inside the /16 range that was set in `fixed-range` in `nova.conf`. Currently, there can only be one network, and this set up would use the max IPs available in a /24. You can choose values that let you use any valid amount that you would like.

OpenStack Compute assumes that the first IP address is your network (like 192.168.0.0), that the 2nd IP is your gateway (192.168.0.1), and that the broadcast is the very last IP in the range you defined (192.168.0.255). You can alter the gateway using the `--gateway` flag when invoking `nova network-create`. You are unlikely to need to modify the network or broadcast addresseses, but if you do, you will need to manually edit the networks table in the database.

## Enabling Access to VMs on the Compute Node

One of the most commonly missed configuration areas is not allowing the proper access to VMs. Use nova client commands to enable access. Below, you will find the commands to allow `ping` and `ssh` to your VMs :



### Note

These commands need to be run as root only if the credentials used to interact with `nova-api` have been put under `/root/.bashrc`. If the EC2 credentials have been put into another user's `.bashrc` file, then, it is necessary to run these commands as the user.

```
$ 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
```

Another common issue is you cannot ping or SSH to your instances after issuing the `euca-authorize` commands. Something to look at is the amount of `dnsmasq` processes that are running. If you have a running instance, check to see that TWO `dnsmasq` processes are running. If not, perform the following:

```
$ sudo killall dnsmasq
```

```
$ sudo service nova-network restart
```

If you get the instance not found message while performing the restart, that means the service was not previously running. You simply need to start it instead of restarting it:

```
$ sudo service nova-network start
```

## Configuring Multiple Compute Nodes

If your goal is to split your VM load across more than one server, you can connect an additional **nova-compute** node to a cloud controller node. This configuring can be reproduced on multiple compute servers to start building a true multi-node OpenStack Compute cluster.

To build out and scale the Compute platform, you spread out services amongst many servers. While there are additional ways to accomplish the build-out, this section describes adding compute nodes, and the service we are scaling out is called **nova-compute**.

For a multi-node install you only make changes to `nova.conf` and copy it to additional compute nodes. Ensure each `nova.conf` file points to the correct IP addresses for the respective services.

By default, Nova sets the bridge device based on the setting in `flat_network_bridge`. Now you can edit `/etc/network/interfaces` with the following template, updated with your IP information.

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

# The primary network interface
auto br100
iface br100 inet static
    bridge_ports     eth0
    bridge_stp      off
    bridge_maxwait   0
    bridge_fd       0
    address xxx.xxx.xxx.xxx
    netmask xxx.xxx.xxx.xxx
    network xxx.xxx.xxx.xxx
    broadcast xxx.xxx.xxx.xxx
    gateway xxx.xxx.xxx.xxx
    # dns-* options are implemented by the resolvconf package, if installed
    dns-nameservers xxx.xxx.xxx.xxx
```

Restart networking:

```
$ sudo service networking restart
```

With `nova.conf` updated and networking set, configuration is nearly complete. First, bounce the relevant services to take the latest updates:

```
$ sudo service libvirtd restart
$ sudo service nova-compute restart
```

To avoid issues with KVM and permissions with Nova, run the following commands to ensure we have VM's that are running optimally:

```
# chgrp kvm /dev/kvm
# chmod g+rwx /dev/kvm
```

If you want to use the 10.04 Ubuntu Enterprise Cloud images that are readily available at <http://uec-images.ubuntu.com/releases/10.04/release/>, you may run into delays with booting. Any server that does not have **nova-api** running on it needs this iptables entry so that UEC images can get metadata info. On compute nodes, configure the iptables with this next step:

```
# iptables -t nat -A PREROUTING -d 169.254.169.254/32 -p tcp -m tcp --dport 80
-j DNAT --to-destination $NOVA_API_IP:8773
```

Lastly, confirm that your compute node is talking to your cloud controller. From the cloud controller, run this database query:

```
$ mysql -u$MYSQL_USER -p$MYSQL_PASS nova -e 'select * from services;'
```

In return, you should see something similar to this:

created_at	updated_at	deleted_at	deleted	host	binary	topic	report_count	disabled	availability_zone
2011-01-28 22:52:46	2011-02-03 06:55:48	NULL	0	osdemo02	nova-network	network	46064	0	1
2011-01-28 22:52:48	2011-02-03 06:55:57	NULL	0	osdemo02	nova-compute	compute	46056	0	2
2011-01-28 22:52:52	2011-02-03 06:55:50	NULL	0	osdemo02	nova-scheduler	scheduler	46065	0	3
2011-01-29 23:49:29	2011-02-03 06:54:26	NULL	0	osdemo01	nova-compute	compute	37050	0	4
2011-01-30 23:42:24	2011-02-03 06:55:44	NULL	0	osdemo04	nova-compute	compute	28484	0	9
2011-01-30 21:27:28	2011-02-03 06:54:23	NULL	0	osdemo05	nova-compute	compute	29284	0	8

You can see that osdemo0{1,2,4,5} are all running **nova-compute**. When you start spinning up instances, they will allocate on any node that is running **nova-compute** from this list.

## Determining the Version of Compute

You can find the version of the installation by using the **nova-manage** command:

```
$ nova-manage version list
```

## Diagnose your compute nodes

You can obtain extra informations about the running virtual machines: their CPU usage, the memory, the disk IO or network IO, per instance, by running the **nova diagnostics** command with a server ID:

```
$ nova diagnostics <serverID>
```

The output of this command will vary depending on the hypervisor. Example output when the hypervisor is Xen:

Property	Value
cpu0	4.3627
memory	1171088064.0000
memory_target	1171088064.0000
vbd_xvda_read	0.0
vbd_xvda_write	0.0
vif_0_rx	3223.6870
vif_0_tx	0.0
vif_1_rx	104.4955
vif_1_tx	0.0

While the command should work with any hypervisor that is controlled through libvirt (e.g., KVM, QEMU, LXC), it has only been tested with KVM. Example output when the hypervisor is KVM:

Property	Value
cpu0_time	2870000000
memory	524288
vda_errors	-1
vda_read	262144
vda_read_req	112
vda_write	5606400
vda_write_req	376
vnet0_rx	63343
vnet0_rx_drop	0
vnet0_rx_errors	0
vnet0_rx_packets	431
vnet0_tx	4905
vnet0_tx_drop	0
vnet0_tx_errors	0
vnet0_tx_packets	45

## General Compute Configuration Overview

Most configuration information is available in the `nova.conf` configuration option file. Here are some general purpose configuration options that you can use to learn more about the configuration option file and the node. The configuration file `nova.conf` is typically stored in `/etc/nova/nova.conf`.

You can use a particular configuration option file by using the option (`nova.conf`) parameter when running one of the `nova-*` services. This inserts configuration option

definitions from the given configuration file name, which may be useful for debugging or performance tuning. Here are some general purpose configuration options.

If you want to maintain the state of all the services, you can use the `state_path` configuration option to indicate a top-level directory for storing data related to the state of Compute including images if you are using the Compute object store.

## Example `nova.conf` Configuration Files

The following sections describe many of the configuration option settings that can go into the `nova.conf` files. Copies of each `nova.conf` file need to be copied to each compute node. Here are some sample `nova.conf` files that offer examples of specific configurations.

## KVM, Flat, MySQL, and Glance, OpenStack or EC2 API

This example `nova.conf` file is from an internal Rackspace test system used for demonstrations.

```
[DEFAULT]

# LOGS/STATE
verbose=True
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
rootwrap_config=/etc/nova/rootwrap.conf

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
# configured in cinder.conf

# DATABASE
sql_connection=mysql://nova:yourpassword@192.168.206.130/nova

# COMPUTE
libvirt_type=qemu
compute_driver=libvirt.LibvirtDriver
instance_name_template=instance-%08x
api_paste_config=/etc/nova/api-paste.ini

# COMPUTE/APIS: if you have separate configs for separate services
# this flag is required for both nova-api and nova-compute
allow_resize_to_same_host=True

# APIS
osapi_compute_extension=nova.api.openstack.compute.contrib.standard_extensions
ec2_dmz_host=192.168.206.130
s3_host=192.168.206.130

# RABBITMQ
rabbit_host=192.168.206.130

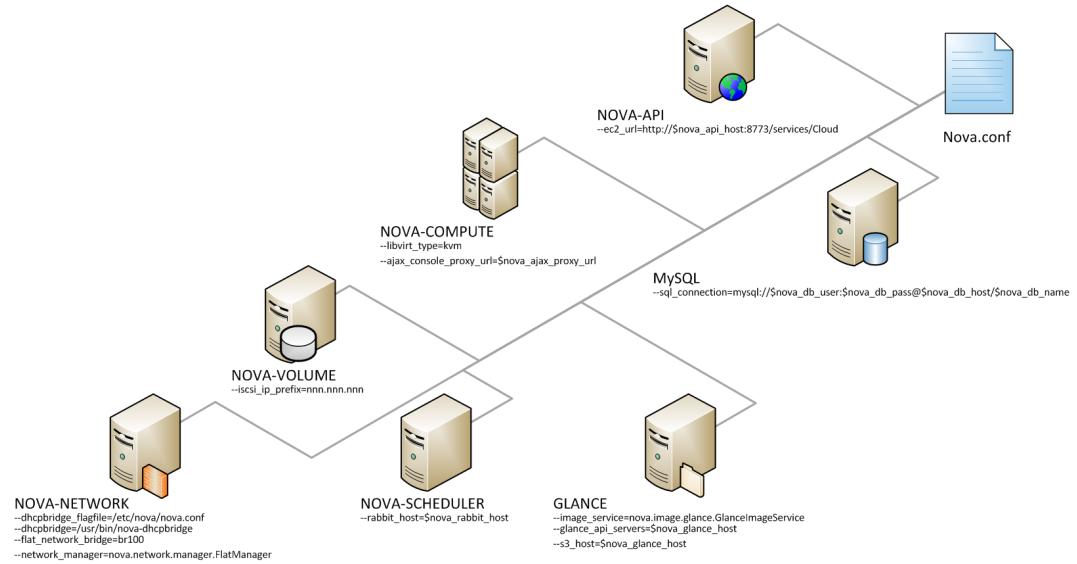
# GLANCE
```

```
image_service=nova.image.glance.GlanceImageService
glance_api_servers=192.168.206.130:9292

# NETWORK
network_manager=nova.network.manager.FlatDHCPManager
force_dhcp_release=True
dhcpbridge_flagfile=/etc/nova/nova.conf
firewall_driver=nova.virt.libvirt.firewall.IptablesFirewallDriver
# Change my_ip to match each host
my_ip=192.168.206.130
public_interface=eth0
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=''

# NOVNC CONSOLE
novncproxy_base_url=http://192.168.206.130:6080/vnc_auto.html
# Change vncserver_proxyclient_address and vncserver_listen to match each
# compute host
vncserver_proxyclient_address=192.168.206.130
vncserver_listen=192.168.206.130

# AUTHENTICATION
auth_strategy=keystone
[keystone_auth]
auth_host = 127.0.0.1
auth_port = 35357
auth_protocol = http
admin_tenant_name = service
admin_user = nova
admin_password = nova
signing_dirname = /tmp/keystone-signing-nova
```

**Figure 4.1. KVM, Flat, MySQL, and Glance, OpenStack or EC2 API**

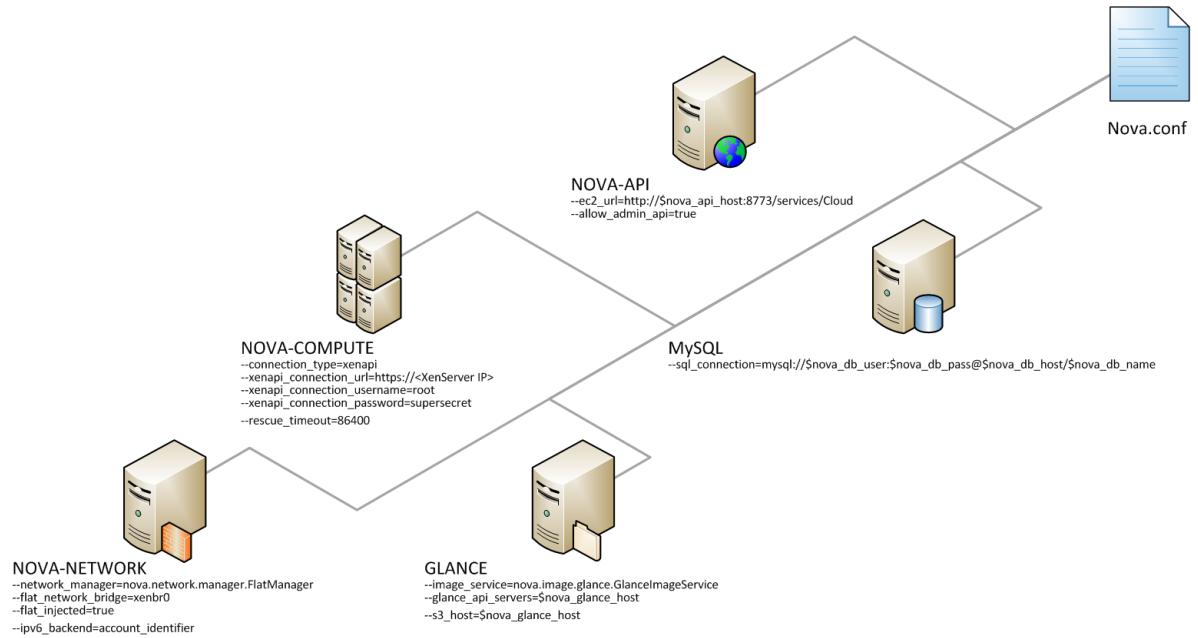
## XenServer, Flat networking, MySQL, and Glance, OpenStack API

This example `nova.conf` file is from an internal Rackspace test system.

```
verbose
nodaemon
sql_connection=mysql://root:<password>@127.0.0.1/nova
network_manager=nova.network.manager.FlatManager
image_service=nova.image.glance.GlanceImageService
flat_network_bridge=xenbr0
compute_driver=xenapi.XenAPIDriver
xenapi_connection_url=https://<XenServer IP>
xenapi_connection_username=root
xenapi_connection_password=supersecret
rescue_timeout=86400
xenapi_inject_image=false
use_ipv6=true

# To enable flat_injected, currently only works on Debian-based systems
flat_injected=true
ipv6_backend=account_identifier
ca_path=./nova/CA

# Add the following to your conf file if you're running on Ubuntu Maverick
xenapi_remap_vbd_dev=true
```

**Figure 4.2. KVM, Flat, MySQL, and Glance, OpenStack or EC2 API**

## Configuring Logging

You can use `nova.conf` configuration options to indicate where Compute will log events, the level of logging, and customize log formats.

To customize log formats for OpenStack Compute, use these configuration option settings.

**Table 4.1. Description of nova.conf log file configuration options**

Configuration option=Default value	(Type) Description
default_log_levels= "amqplib=WARN,sqlalchemy=WARN,boto=WARN,suds=INFO,eventlet.wsgi.server=WARN"	(ListOpt) list of logger=LEVEL pairs
instance_format=[instance: %(uuid)s]	(StrOpt) If an instance is passed with the log message, format it like this
instance_uuid_format=[instance: %(uuid)s]	(StrOpt) If an instance UUID is passed with the log message, format it like this
log_config=<None>	(StrOpt) If this option is specified, the logging configuration file specified is used and overrides any other logging options specified. Please see the Python

Configuration option=Default value	(Type) Description
	logging module documentation for details on logging configuration files.
log_date_format=%Y-%m-%d %H:%M:%S	(StrOpt) Format string for %(asctime)s in log records. Default: %default
log_dir=<None>	(StrOpt) (Optional) The directory to keep log files in (will be prepended to --logfile)
log_file=<None>	(StrOpt) (Optional) Name of log file to output to. If not set, logging will go to stdout.
log_format="%(asctime)s %(levelname)s [%(name)s] %(message)s"	(StrOpt) A logging.Formatter log message format string which may use any of the available logging.LogRecord attributes. Default: %default
logdir=<None>	(StrOpt) Log output to a per-service log file in named directory
logfile=<None>	(StrOpt) Log output to a named file
logfile_mode=0644	(StrOpt) Default file mode used when creating log files
logging_context_format_string="%(asctime)s %(levelname)s %(name)s [%(request_id)s %(user_id)s %(project_id)s] %(instance)s%(message)s"	(StrOpt) format string to use for log messages with context
logging_debug_format_suffix="from (%(pid=%(process)d)%(funcName)s)s (%(pathname)s)s:%(lineno)d"	(StrOpt) data to append to log format when level is DEBUG
logging_default_format_string="%(asctime)s %(levelname)s %(name)s [-] %(instance)s%(message)s"	(StrOpt) format string to use for log messages without context
logging_exception_prefix="%(asctime)s TRACE %(name)s %(instance)s"	(StrOpt) prefix each line of exception output with this format
publish_errors=false	(BoolOpt) publish error events
publish_errors=true	(BoolOpt) publish error events
use_syslog=false	(BoolOpt) Use syslog for logging
syslog_log_facility=LOG_USER	(StrOpt) syslog facility to receive log lines

## Configuring Hypervisors

OpenStack Compute requires a hypervisor and supports several hypervisors and virtualization standards. Configuring and running OpenStack Compute to use a particular hypervisor takes several installation and configuration steps. The `libvirt_type` configuration option indicates which hypervisor will be used. Refer to [Hypervisor Configuration Basics](#) for more details. To customize hypervisor support in OpenStack Compute, refer to these configuration settings in `nova.conf`.

**Table 4.2. Description of `nova.conf` file configuration options for hypervisors**

Configuration option=Default value	(Type) Description
block_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_UNDEFINE_DEST,VIR_MIGRATE_PRESERVE,VIR_MIGRATE_BLOCK,MIGRATION_BLOCK,MIGRATION_BLOCK_INCR	(StringOpt) Define REBORN flag for migration. NON_SHARED_INC
checksum_base_images=false	(BoolOpt) Used as an additional check to detect if cached images have become corrupted. If true, the compute service will write checksums for image files in the <code>/var/lib/nova/instances/_base</code> directory to disk, and do periodic checks to verify that this checksum is valid. If the checksum fails to validate, the failure is recorded to the log as an error, but no other action is taken: it is assumed that an operator will monitor the logs and take appropriate action. If false, image hashes are not verified.
hyperv_attaching_volume_retry_count=10	(IntOpt) Number of times to retry attaching to a volume when using the Hyper-V hypervisor

Configuration option=Default value	(Type) Description
hyperv_wait_between_attach_retry=5	(IntOpt) To be written: found in /nova/virt/hyperv/volumeops.py
libvirt_cpu_mode=<None>	(StrOpt) Configures the guest CPU model exposed to the hypervisor. Valid options are: custom, host-model, host-passthrough, none. If the hypervisor is KVM or QEMU, the default value is host-model, otherwise the default value is none.
libvirt_cpu_model=<None>	(StrOpt) Specify the guest CPU model exposed to the hypervisor. This configuration option is only applicable if libvirt_cpu_mode is set to custom. Valid options: one of the named models specified in /usr/share/libvirt/cpu_map.xml, e.g.: Westmere, Nehalem, Opteron_G3.
libvirt_disk_prefix=<None>	(StrOpt) Override the default disk prefix for the devices attached to a server, which is dependent on libvirt_type. (valid options are: sd, xvd, uvd, vd)
libvirt_inject_key=true	(BoolOpt) Inject the ssh public key at boot time
libvirt_inject_partition=1	(IntOpt) The partition to inject to : -2 => disable, -1 => inspect (libguestfs only), 0 => not partitioned, >0 => partition number'
libvirt_images_type=default	(StrOpt) Instance ephemeral storage backend format. Acceptable values are: raw, qcow2, lvm, default. If default is specified, then use_cow_images flag is used instead of this one. Please note, that current snapshot mechanism in OpenStack Compute works only with instances backed with Qcow2 images.
libvirt_images_volume_group=None	(StrOpt) LVM Volume Group that is used for instance ephemerals, when you specify libvirt_images_type=lvm.
libvirt_inject_password=false	(BoolOpt) Inject the admin password at boot time, without an agent.
libvirt_lvm_snapshot_size=1000	(IntOpt) The amount of storage (in megabytes) to allocate for LVM snapshot copy-on-write blocks.
libvirt_nonblocking=true	(BoolOpt) Use a separated OS thread pool to realize non-blocking libvirt calls
libvirt_snapshots_directory=\$instances_path/snapshots	(StrOpt) Location where libvirt driver will store snapshots before uploading them to image service
libvirt_snapshot_compression=False	(BoolOpt) Compresses snapshot images when possible. This currently applies exclusively to qcow2 images.
libvirt_sparse_logical_volumes=false	(BoolOpt) Create sparse (not fully allocated) LVM volumes for instance ephemerals if you use LVM backend for them.
libvirt_type=kvm	(StrOpt) Libvirt domain type (valid options are: kvm, lxc, qemu, uml, xen)
libvirt_uri=	(StrOpt) Override the default libvirt URI (which is dependent on libvirt_type)
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtGenericVifDriver	(StrOpt) The libvirt VIF driver to configure the VIFs.
libvirt_volume_drivers="iscsi=nova.virt.libvirt.volume.LibvirtIscsiVolumeDriver, local=nova.virt.libvirt.volume.LibvirtVolumeDriver, fake=nova.virt.libvirt.volume.LibvirtFakeVolumeDriver, rbd=nova.virt.libvirt.volume.LibvirtNetVolumeDriver, sheepdog=nova.virt.libvirt.volume.LibvirtNetVolumeDriver, glusterfs=nova.virt.libvirt.volume.LibvirtGlusterfsVolumeDriver"	(StrOpt) libvirt volume drivers for remote volumes.
libvirt_wait_soft_reboot_seconds=120	(IntOpt) Number of seconds to wait for instance to shut down after soft reboot request is made. We fall back to

Configuration option=Default value	(Type) Description
	hard reboot if instance does not shutdown within this window.
limit_cpu_features=false	(BoolOpt) Used by Hyper-V
remove_unused_base_images=true	(BoolOpt) Indicates whether unused base images should be removed
remove_unused_kernels=false	(BoolOpt) Should unused kernel images be removed? If unused images should be removed set to true, if not, set to false. This option is only safe to set to true if all compute nodes have been updated to support this option so that older image cache managers on remote compute nodes are prevented from cleaning up kernels because they appear unused. This will be enabled by default in a future release.
remove_unused_original_minimum_age_seconds=86400	(IntOpt) Unused unresized base images younger than this will not be removed
remove_unused_resized_minimum_age_seconds=3600	(IntOpt) Unused resized base images younger than this will not be removed
rescue_image_id=<None>	(StrOpt) Rescue ami image
rescue_kernel_id=<None>	(StrOpt) Rescue aki image
rescue_ramdisk_id=<None>	(StrOpt) Rescue ari image
snapshot_image_format=<None>	(StrOpt) Snapshot image format (valid options are : raw, qcow2, vmdk, vdi). Defaults to same as source image
use_usb_tablet=true	(BoolOpt) Sync virtual and real mouse cursors in Windows VMs
libvirt integration	
libvirt_ovs_bridge=br-int	(StrOpt) Name of Integration Bridge used by Open vSwitch
libvirt_use_virtio_for_bridges=false	(BoolOpt) Use virtio for bridge interfaces
VMWare integration	
vmwareapi_wsdl_loc=<None>	(StrOpt) VIM Service WSDL Location e.g http://<server>/vimService.wsdl, due to a bug in vSphere ESX 4.1 default wsdl.
vmware_vif_driver=nova.virt.vmwareapi.vif.VMWareVlanBridgeDriver	(Type Opt) The VMWare VIF driver to configure the VIFs.
vmwareapi_api_retry_count=10	(FloatOpt) The number of times we retry on failures, e.g., socket error, etc. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_ip=<None>	(StrOpt) URL for connection to VMWare ESX host. Required if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_password=<None>	(StrOpt) Password for connection to VMWare ESX host. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_username=<None>	(StrOpt) Username for connection to VMWare ESX host. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_task_poll_interval=5.0	(FloatOpt) The interval used for polling of remote tasks. Used only if compute_driver is vmwareapi.VMwareESXDriver,
vmwareapi_vlan_interface=vmnic0	(StrOpt) Physical ethernet adapter name for vlan networking
powervm_mgr_type=ivm	(StrOpt) PowerVM system manager type (ivm, hmc)
powervm_mgr=<None>	(StrOpt) PowerVM manager host or ip
powervm_vios=powervm_mgr	(StrOpt) PowerVM VIOS host or ip if different from manager

Configuration option=Default value	(Type) Description
powervm_mgr_user=<None>	(StrOpt) PowerVM manager user name
powervm_mgr_passwd=<None>	(StrOpt) PowerVM manager user password
powervm_img_remote_path=<None>	(StrOpt) PowerVM image remote path. Used to copy and store images from Glance on the PowerVM VIOS LPAR.
powervm_img_local_path=<None>	(StrOpt) Local directory on the compute host to download glance images to.

## Configuring Authentication and Authorization

There are different methods of authentication for the OpenStack Compute project, including no authentication. The preferred system is the OpenStack Identity Service, code-named Keystone. Refer to [Identity Management](#) for additional information.

To customize authorization settings for Compute, see these configuration settings in `nova.conf`.

**Table 4.3. Description of `nova.conf` configuration options for authentication**

Configuration option=Default value	(Type) Description
auth_strategy=noauth	(StrOpt) The strategy to use for authentication. Supports noauth or keystone.
auth_token_ttl=3600	(IntOpt) Seconds for auth tokens to linger
ldap_cloudadmin=cn=cloudadmins,ou=Groups,dc=example,dc=com	(StrOpt) cn for Cloud Admins
ldap_developer=cn=developers,ou=Groups,dc=example,dc=com	(StrOpt) cn for Developers
ldap_itsec=cn=itsec,ou=Groups,dc=example,dc=com	(StrOpt) cn for ItSec
ldap_netadmin=cn=netadmins,ou=Groups,dc=example,dc=com	(StrOpt) cn for NetAdmins
ldap_password=changeme	(StrOpt) LDAP password
ldap_project_subtree=ou=Groups,dc=example,dc=com	(StrOpt) OU for Projects
ldap_schema_version=2	(IntOpt) Current version of the LDAP schema
ldap_sysadmin=cn=sysadmins,ou=Groups,dc=example,dc=com	(StrOpt) cn for Sysadmins
ldap_url=ldap://localhost	(StrOpt) Point this at your ldap server
ldap_user_dn=cn=Manager,dc=example,dc=com	(StrOpt) DN of admin user
ldap_user_id_attribute=uid	(StrOpt) Attribute to use as id
ldap_user_modify_only=false	(BoolOpt) Modify user attributes instead of creating/deleting
ldap_user_name_attribute=cn	(StrOpt) Attribute to use as name
ldap_user_subtree=ou=Users,dc=example,dc=com	(StrOpt) OU for Users
ldap_user_unit=Users	(StrOpt) OID for Users
role_project_subtree=ou=Groups,dc=example,dc=com	(StrOpt) OU for Roles
allowed_roles=cloudadmin,itsec,sysadmin,netadmin,developer	(ListOpt) Allowed roles for project
auth_driver=nova.auth.dbdriver.DbDriver	(StrOpt) Driver that auth manager uses
credential_cert_file=cert.pem	(StrOpt) Filename of certificate in credentials zip
credential_key_file=pk.pem	(StrOpt) Filename of private key in credentials zip
credential_rc_file=%src	(StrOpt) Filename of rc in credentials zip %s will be replaced by name of the region (nova by default)
credential_vpn_file=nova-vpn.conf	(StrOpt) Filename of certificate in credentials zip
credentials_template=\$pybasedir/nova/auth/novarc.template	(StrOpt) Template for creating users rc file

Configuration option=Default value	(Type) Description
global_roles=cloudadmin,itsec	(ListOpt) Roles that apply to all projects
superuser_roles=cloudadmin	(ListOpt) Roles that ignore authorization checking completely
vpn_client_template=\$pybasedir/nova/cloudpipe/client.ovpn.template	(StrOpt) Template for creating users VPN file

To customize certificate authority settings for Compute, see these configuration settings in `nova.conf`.

**Table 4.4. Description of nova.conf file configuration options for credentials (crypto)**

Configuration option=Default value	(Type) Description
ca_file=cacert.pem	(StrOpt) Filename of root CA (Certificate Authority)
ca_path=\$state_path/CA	(StrOpt) Where we keep our root CA
crl_file=crl.pem	(StrOpt) Filename of root Certificate Revocation List
key_file=private/cakey.pem	(StrOpt) Filename of private key
keys_path=\$state_path/keys	(StrOpt) Where we keep our keys
project_cert_subject="/C=US/ST=California/O=OpenStack/OU=NovaDev/CN=project-ca.%16s-%s"	(StrOpt) Subject for certificate for projects, %s for project, timestamp
use_project_ca=false	(BoolOpt) Whether to use a CA for each project (tenant)
user_cert_subject="/C=US/ST=California/O=OpenStack/OU=NovaDev/CN=%16s-%16s-%s"	(StrOpt) Subject for certificate for users, %s for project, user, timestamp

To customize Compute and the Identity service to use LDAP as a backend, refer to these configuration settings in `nova.conf`.

**Table 4.5. Description of nova.conf file configuration options for LDAP**

Configuration option=Default value	(Type) Description
ldap_cloudadmin="cn=cloudadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Cloud Admins
ldap_developer="cn=developers,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Developers
ldap_itsec="cn=itsec,ou=Groups,dc=example,dc=com"	(StrOpt) CN for ItSec
ldap_netadmin="cn=netadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for NetAdmins
ldap_password="changeme"	(StrOpt) LDAP password
ldap_suffix="cn=example,cn=com"	(StrOpt) LDAP suffix
ldap_use_dumb_member=False	(BoolOpt) Simulates an LDAP member
ldap_project_subtree="ou=Groups,dc=example,dc=com"	(StrOpt) OU for Projects
ldap_objectClass=inetOrgPerson	(StrOpt) LDAP objectClass to use
ldap_schema_version=2	(IntOpt) Current version of the LDAP schema
ldap_sysadmin="cn=sysadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Sysadmins
ldap_url="ldap://localhost"	(StrOpt) Point this at your ldap server
ldap_user="dc=Manager,dc=example,dc=com"	(StrOpt) LDAP User
ldap_user_tree_dn="ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
ldap_user_dn="cn=Manager,dc=example,dc=com"	(StrOpt) DN of Users

Configuration option=Default value	(Type) Description
ldap_user_objectClass= inetOrgPerson	(StrOpt) DN of Users
ldap_user_id_attribute= cn	(StrOpt) Attribute to use as id
ldap_user_modify_only=false	(BoolOpt) Modify user attributes instead of creating/deleting
ldap_user_name_attribute= cn	(StrOpt) Attribute to use as name
ldap_user_subtree= "ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
ldap_user_unit= "Users"	(StrOpt) OID for Users
ldap_tenant_tree_dn="ou=Groups,dc=example,dc=com"	(StrOpt) OU for Tenants
ldap_tenant_objectclass= groupOfNames	(StrOpt) LDAP ObjectClass to use for Tenants
ldap_tenant_id_attribute= cn	(strOpt) Attribute to use as Tenant
ldap_tenant_member_attribute= member	(strOpt) Attribute to use as Member
ldap_role_tree_dn= "ou=Roles,dc=example,dc=com"	(strOpt) OU for Roles
ldap_role_objectclass= organizationalRole	(strOpt) LDAP ObjectClass to use for Roles
ldap_role_project_subtree= "ou=Groups,dc=example,dc=com"	(StrOpt) OU for Roles
ldap_role_member_attribute= roleOccupant	(StrOpt) Attribute to use as Role member
ldap_role_id_attribute= cn	(StrOpt) Attribute to use as Role

## Configuring Compute to use IPv6 Addresses

You can configure Compute to use both IPv4 and IPv6 addresses for communication by putting it into a IPv4/IPv6 dual stack mode. In IPv4/IPv6 dual stack mode, instances can acquire their IPv6 global unicast address by stateless address autoconfiguration mechanism [RFC 4862/2462]. IPv4/IPv6 dual stack mode works with VlanManager and FlatDHCPManager networking modes. In VlanManager, different 64bit global routing prefix is used for each project. In FlatDHCPManager, one 64bit global routing prefix is used for all instances.

This configuration has been tested with VM images that have IPv6 stateless address autoconfiguration capability (must use EUI-64 address for stateless address autoconfiguration), a requirement for any VM you want to run with an IPv6 address. Each node that executes a nova-\* service must have python-netaddr and radvd installed.

On all nova-nodes, install python-netaddr:

```
$ sudo apt-get install -y python-netaddr
```

On all nova-network nodes install radvd and configure IPv6 networking:

```
$ sudo apt-get install -y radvd
$ sudo bash -c "echo 1 > /proc/sys/net/ipv6/conf/all/forwarding"
$ sudo bash -c "echo 0 > /proc/sys/net/ipv6/conf/all/accept_ra"
```

Edit the nova.conf file on all nodes to set the use\_ipv6 configuration option to True. Restart all nova-services.

When using the command **nova network-create** you can add a fixed range for IPv6 addresses. You must specify public or private after the create parameter.

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

You can set IPv6 global routing prefix by using the `--fixed_range_v6` parameter. The default is: `fd00::/48`. When you use `FlatDHCPManager`, the command uses the original value of `--fixed_range_v6`. When you use `VlanManager`, the command creates prefixes of subnet by incrementing subnet id. Guest VMs uses this prefix for generating their IPv6 global unicast address.

Here is a usage example for `VlanManager`:

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

Here is a usage example for `FlatDHCPManager`:

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

**Table 4.6. Description of nova.conf configuration options for IPv6**

Configuration option=Default value	(Type) Description
<code>fixed_range_v6=fd00::/48</code>	(StrOpt) Fixed IPv6 address block
<code>gateway_v6=&lt;None&gt;</code>	(StrOpt) Default IPv6 gateway
<code>ipv6_backend=rfc2462</code>	(StrOpt) Backend to use for IPv6 generation
<code>use_ipv6=false</code>	(BoolOpt) use IPv6

## Configuring Image Service and Storage for Compute

Compute relies on an external image service to store virtual machine images and maintain a catalog of available images. Compute is configured by default to use the OpenStack Image service (Glance), which is the only currently supported image service.

If your installation requires the use of euca2ools for registering new images, you will need to run the `nova-objectstore` service. This service provides an Amazon S3 frontend for Glance, which is needed because euca2ools can only upload images to an S3-compatible image store.

**Table 4.7. Description of nova.conf file configuration options for S3 access to image storage**

Configuration option=Default value	(Type) Description
<code>image_decryption_dir=/tmp</code>	(StrOpt) parent dir for tempdir used for image decryption
<code>s3_access_key=notchecked</code>	(StrOpt) access key to use for s3 server for images
<code>s3_affix_tenant=false</code>	(BoolOpt) whether to affix the tenant id to the access key when downloading from s3
<code>s3_secret_key=notchecked</code>	(StrOpt) secret key to use for s3 server for images
<code>s3_use_ssl=false</code>	(BoolOpt) whether to use ssl when talking to s3

## Configuring Migrations



### Note

This feature is for cloud administrators only.

Migration allows 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.

There are two types of migration:

- **Migration** (or non-live migration): In this case the instance will be shut down (and the instance will know that it has been rebooted) for a period of time in order to be moved to another hypervisor.
- **Live migration** (or true live migration): Almost no instance downtime, it is useful when the instances must be kept running during the migration.

There are two types of **live migration**:

- **Shared storage based live migration**: In this case both hypervisors have access to a shared storage.
- **Block live migration**: for this type of migration, no shared storage is required.

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

## KVM-Libvirt

### Prerequisites

- **Hypervisor**: KVM with libvirt
- **Shared storage**: `NOVA-INST-DIR/instances/` (eg `/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

Migrations done by the Compute service do not use libvirt's live migration functionality by default. Because of this, guests are suspended before migration and may therefore experience several minutes of downtime. See [True Migration for KVM and Libvirt](#) for more details.



### Note

This guide assumes the default value for `instances_path` in your `nova.conf` (`NOVA-INST-DIR/instances`). If you have changed the `state_path` or `instances_path` variables, please modify accordingly.



### Note

You must specify `vncserver_listen=0.0.0.0` or live migration will not work correctly.

### Example Nova Installation Environment

- Prepare 3 servers at least; for example, HostA, HostB and HostC
- HostA is the "Cloud Controller", and should be running: nova-api, nova-scheduler, nova-network, cinder-volume, nova-objectstore.
- HostB and HostC are the "compute nodes", running nova-compute.
- Ensure that, *NOVA-INST-DIR* (set with state\_path in nova.conf) is same on all hosts.
- In this example, HostA will be the NFSv4 server which exports *NOVA-INST-DIR/instances*, and HostB and HostC mount it.

### System configuration

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 nova and libvirt users are identical between each of your servers. This ensures that the permissions on the NFS mount will work correctly.
3. Follow the instructions at [the Ubuntu NFS HowTo to setup an NFS server on HostA, and NFS Clients on HostB and HostC](#).

Our aim is to export *NOVA-INST-DIR/instances* from HostA, and have it readable and writable by the nova user on HostB and HostC.

4. Using your knowledge from the Ubuntu documentation, configure the NFS server at HostA by adding a line to /etc(exports

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

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

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

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

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

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

6. Configure NFS at HostB and HostC by adding below to /etc/fstab.

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

Then ensure that the exported directory can be mounted.

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

Check that "*NOVA-INST-DIR/instances/*" directory can be seen at HostA

```
$ ls -ld NOVA-INST-DIR/instances/  
  
drwxr-xr-x 2 nova nova 4096 2012-05-19 14:34 nova-install-dir/instances/
```

Perform the same check at HostB and HostC - paying special attention to the permissions (nova 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/sdal        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. Modify */etc/libvirt/libvirtd.conf*:

```
before : #listen_tls = 0  
after : listen_tls = 0  
  
before : #listen_tcp = 1  
after : listen_tcp = 1  
  
add: auth_tcp = "none"
```

Modify */etc/init/libvirt-bin.conf*

```
before : exec /usr/sbin/libvirtd -d  
after : exec /usr/sbin/libvirtd -d -l
```

Modify */etc/default/libvirt-bin*

```
before :libvirtd_opts=" -d"  
after :libvirtd_opts=" -d -l"
```

Restart libvirt. After executing 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.

Information about ports used with libvirt can be found at [the libvirt documentation](#). By default, libvirt listens on TCP port 16509 and an ephemeral TCP range from 49152 to 49261 is used for the KVM communications. As this guide has disabled libvirt auth, you should take good care that these ports are only open to hosts within your installation.

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

**Table 4.8. Description of nova.conf file configuration options for live migration**

Configuration option=Default value	(Type) Description
live_migration_bandwidth=0	(IntOpt) Maximum bandwidth to be used during migration transfer, in Mbps.
live_migration_flag= VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER	(StrOpt) Migration flags to be set for live migration, defines parameters for use during the migration.
live_migration_retry_count=30	(IntOpt) Number of one-second retries needed in live_migration.
live_migration_uri=qemu+tcp:///%s/system	(StrOpt) Define Host URI used by live_migration feature. Any included "%s" is replaced with the migration target host name.

## Enabling true live migration

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

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

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

## XenServer

### Shared Storage

#### Prerequisites

- **Compatible XenServer hypervisors.** For more information, please refer to the [Requirements for Creating Resource Pools](#) section of the XenServer Administrator's Guide.
- **Shared storage:** an NFS export, visible to all XenServer hosts.



## Note

Please check the [NFS VHD](#) section of the XenServer Administrator's Guide for the supported NFS versions.

In order to use shared storage live migration with XenServer hypervisors, the hosts must be joined to a XenServer pool. In order to create that pool, a host aggregate must be created with special metadata. This metadata will be used by the XAPI plugins to establish the pool.

1. Add an NFS VHD storage to your master XenServer, and set it as default SR. For more information, please refer to the [NFS VHD](#) section of the XenServer Administrator's Guide.
2. Configure all the compute nodes to use the default sr for pool operations, by including:

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

in your `nova.conf` configuration files across your compute nodes.

3. Create a host aggregate

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

The command will display a table which contains the id of the newly created aggregate. Now add special metadata to the aggregate, to mark it as a hypervisor pool

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

Make the first compute node part of that aggregate

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

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

4. Add additional hosts to the pool:

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



## Note

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

## Block migration

### Prerequisites

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

**Note**

Please note, that you need to use an extra option `--block-migrate` for the live migration command, in order to use block migration.

**Note**

Please note, that block migration works only with EXT local storage SRs, and the server should not have any volumes attached.

## Configuring Resize

Resize (or Server resize) is the ability to change the flavor of a server, thus allowing it to upscale or downscale according to user needs. In order for this feature to work properly, some underlying virt layers may need further configuration; this section describes the required configuration steps for each hypervisor layer provided by OpenStack.

### XenServer

To get resize to work with XenServer (and XCP), please refer to the [Dom0 Modifications for Resize/Migration Support](#) section.

## Installing MooseFS as shared storage for the instances directory

In the previous section we presented a convenient way to deploy a shared storage using NFS. For better transactions performance, you could deploy MooseFS instead.

MooseFS (Moose File System) is a shared file system ; it implements the same rough concepts of shared storage solutions - such as Ceph, Lustre or even GlusterFS.

### Main concepts

- A metadata server (MDS), also called master server, which manages the file repartition, their access and the namespace.
- A metalogger server (MLS) which backs up the MDS logs, including, objects, chunks, sessions and object metadata
- A chunk server (CSS) which store the data as chunks and replicate them across the chunkservers
- A client, which talks with the MDS and interact with the CSS. MooseFS clients manage MooseFS filesystem using FUSE

For more informations, please see the [Official project website](#)

Our setup will be made the following way :

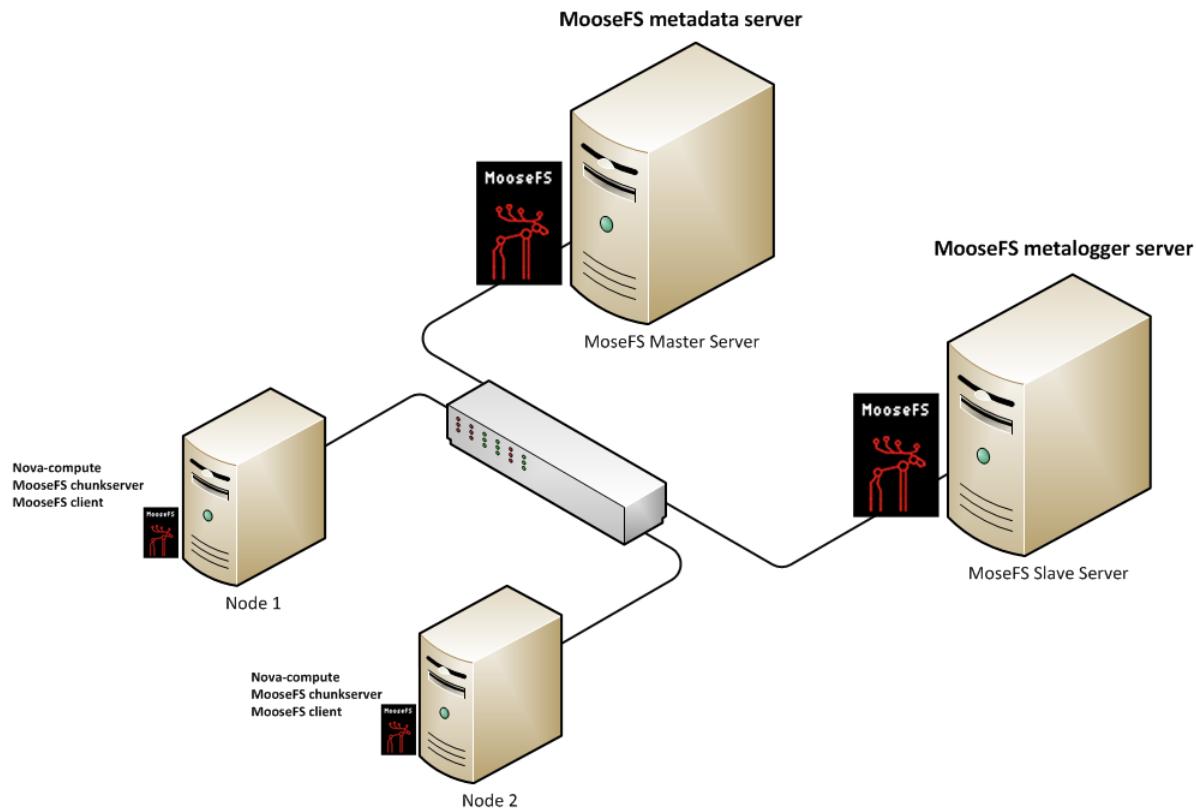
- Two compute nodes running both MooseFS chunkserver and client services.

- One MooseFS master server, running the metadata service.
- One MooseFS slave server, running the metalogger service.

For that particular walkthrough, we will use the following network schema :

- 10.0.10.15 for the MooseFS metadata server admin IP
- 10.0.10.16 for the MooseFS metadata server main IP
- 10.0.10.17 for the MooseFS metalogger server admin IP
- 10.0.10.18 for the MooseFS metalogger server main IP
- 10.0.10.19 for the MooseFS first chunkserver IP
- 10.0.10.20 for the MooseFS second chunkserver IP

**Figure 4.3. MooseFS deployment for OpenStack**



## Installing the MooseFS metadata and metalogger servers

Both components could be run anywhere , as long as the MooseFS chunkservers can reach the MooseFS master server.

In our deployment, both MooseFS master and slave run their services inside a virtual machine ; you just need to make sure to allocate enough memory to the MooseFS metadata server, all the metadata being stored in RAM when the service runs.

## 1. Hosts entry configuration

In the /etc/hosts add the following entry :

```
10.0.10.16    mfsmaster
```

## 2. Required packages

Install the required packages by running the following commands :

```
$ apt-get install zlib1g-dev python pkg-config  
$ yum install make automake gcc gcc-c++ kernel-devel python26 pkg-config
```

## 3. User and group creation

Create the adequate user and group :

```
$ groupadd mfs && useradd -g mfs mfs
```

## 4. Download the sources

Go to the [MooseFS download page](#) and fill the download form in order to obtain your URL for the package.

## 5. Extract and configure the sources

Extract the package and compile it :

```
$ tar -zxvf mfs-1.6.25.tar.gz && cd mfs-1.6.25
```

For the MooseFS master server installation, we disable from the compilation the mfschunkserver and mfsmount components :

```
$ ./configure --prefix=/usr --sysconfdir=/etc/moosefs --localstatedir=/var/lib --with-default-user=mfs --with-default-group=mfs --disable-mfschunkserver --disable-mfsmount  
$ make && make install
```

## 6. Create configuration files

We will keep the default settings, for tuning performance, you can read the [MooseFS official FAQ](#)

```
$ cd /etc/moosefs  
$ cp mfsmaster.cfg.dist mfsmaster.cfg  
$ cp mfsmetalogger.cfg.dist mfsmetalogger.cfg  
$ cp mfsexports.cfg.dist mfsexports.cfg
```

In `/etc/moosefs/mfsexports.cfg` edit the second line in order to restrict the access to our private network :

```
10.0.10.0/24          /          rw,alldirs,maproot=0
```

Create the metadata file :

```
$ cd /var/lib/mfs && cp metadata.mfs.empty metadata.mfs
```

## 7. Power up the MooseFS mfsmaster service

You can now start the `mfsmaster` and `mfscgiserv` deamons on the MooseFS metadataserver (The `mfscgiserv` is a webserver which allows you to see via a web interface the MooseFS status realtime) :

```
$ /usr/sbin/mfsmaster start && /usr/sbin/mfscgiserv start
```

Open the following url in your browser : `http://10.0.10.16:9425` to see the MooseFS status page

## 8. Power up the MooseFS metalogger service

```
$ /usr/sbin/mfsmetalogger start
```

# Installing the MooseFS chunk and client services

In the first part, we will install the last version of FUSE, and proceed to the installation of the MooseFS chunk and client in the second part.

## Installing FUSE

### 1. Required package

```
$ apt-get install util-linux
```

```
$ yum install util-linux
```

### 2. Download the sources and configure them

For that setup we will retrieve the last version of fuse to make sure every function will be available :

```
$ wget http://downloads.sourceforge.net/project/fuse/fuse-2.X/2.9.1/fuse-2.9.1.tar.gz && tar -zxvf fuse-2.9.1.tar.gz && cd fuse-2.9.1
```

```
$ ./configure && make && make install
```

## Installing the MooseFS chunk and client services

For installing both services, you can follow the same steps that were presented before (Steps 1 to 4) :

1. Hosts entry configuration
2. Required packages
3. User and group creation
4. Download the sources
5. Extract and configure the sources

Extract the package and compile it :

```
$ tar -zxvf mfs-1.6.25.tar.gz && cd mfs-1.6.25
```

For the MooseFS chunk server installation, we only disable from the compilation the mfsmaster component :

```
$ ./configure --prefix=/usr --sysconfdir=/etc/moosefs --localstatedir=/var/lib --with-default-user=mfs --with-default-group=mfs --disable-mfsmaster  
$ make && make install
```

6. Create configuration files

The chunk servers configuration is relatively easy to setup. You only need to create on every server directories that will be used for storing the datas of your cluster.

```
$ cd /etc/moosefs  
$ cp mfschunkserver.cfg.dist mfschunkserver.cfg  
$ cp mfshdd.cfg.dist mfshdd.cfg  
$ mkdir /mnt/mfschunks{1,2} && chown -R mfs:mfs /mnt/mfschunks{1,2}
```

Edit /etc/moosefs/mfhdd.cfg and add the directories you created to make them part of the cluster :

```
# mount points of HDD drives  
#  
#/mnt/hd1  
#/mnt/hd2  
#etc.  
  
/mnt/mfschunks1  
/mnt/mfschunks2
```

7. Power up the MooseFS mfschunkserver service

```
$ /usr/sbin/mfschunkserver start
```

## Access to your cluster storage

You can now access your cluster space from the compute node, (both acting as chunkservers) :

```
$ mfsmount /var/lib/nova/instances -H mfsmaster
```

```
mfsmaster accepted connection with parameters: read-
write,restricted_ip ; root mapped to root:root
```

```
$ mount
```

```
/dev/cciss/c0d0p1 on / type ext4 (rw,errors=remount-ro)
proc on /proc type proc (rw,noexec,nosuid,nodev)
none on /sys type sysfs (rw,noexec,nosuid,nodev)
fusectl on /sys/fs/fuse/connections type fusectl (rw)
none on /sys/kernel/debug type debugfs (rw)
none on /sys/kernel/security type securityfs (rw)
none on /dev type devtmpfs (rw,mode=0755)
none on /dev/pts type devpts (rw,noexec,nosuid,gid=5,mode=0620)
none on /dev/shm type tmpfs (rw,nosuid,nodev)
none on /var/run type tmpfs (rw,nosuid,mode=0755)
none on /var/lock type tmpfs (rw,noexec,nosuid,nodev)
none on /var/lib/ureadahead/debugfs type debugfs (rw,relatime)
mfsmaster:9421 on /var/lib/nova/instances type fuse.mfs (rw,allow_other,
default_permissions)
```

You can interact with it the way you would interact with a classical mount, using build-in linux commands (cp, rm, etc...).

The MooseFS client has several tools for managing the objects within the cluster (set replication goals, etc..). You can see the list of the available tools by running

```
$ mfs <TAB> <TAB>
```

```
mfappendchunks    mfschunkserver    mfsfileinfo      mfsgetgoal
mfsmount         mfsrsetgoal       mfssetgoal       mfstools
mfsckiserv       mfsdeleattr      mfsfilerepair   mfsgettrashtime
mfsrgetgoal      mfsrsettrashtime mfssettrashtime
mfscheckfile     mfsdirinfo       mfsgeteattr     mfsmakesnapshot
mfsrgettrashtime mfsseteattr     mfssnapshot
```

You can read the manual for every command. You can also see the [online help](#)

### Add an entry into the fstab file

In order to make sure to have the storage mounted, you can add an entry into the /etc/fstab on both compute nodes :

```
mfsmount  /var/lib/nova/instances  fuse  mfsmaster=mfsmaster,_netdev  0  0
```

## Configuring Database Connections

You can configure OpenStack Compute to use any SQLAlchemy-compatible database. The database name is nova and entries to it are mostly written by the nova-scheduler service, although all the services need to be able to update entries in the database. Use these settings to configure the connection string for the nova database.

**Table 4.9. Description of nova.conf configuration options for databases**

Configuration option=Default value	(Type) Description
db_backend=sqlalchemy	(StrOpt) The backend to use for db
db_driver=nova.db	(StrOpt) driver to use for database access
sql_connection=sqlite:///\$state_path/\$sqlite_db	(StrOpt) The SQLAlchemy connection string used to connect to the database
sql_connection_debug=0	(IntOpt) Verbosity of SQL debugging information. 0=None, 100=Everything
sql_connection_trace=false	(BoolOpt) Add python stack traces to SQL as comment strings
sql_idle_timeout=3600	(IntOpt) timeout before idle sql connections are reaped
sql_max_retries=10	(IntOpt) maximum db connection retries during startup. (setting -1 implies an infinite retry count)
sql_retry_interval=10	(IntOpt) interval between retries of opening a sql connection
sqlite_clean_db=clean.sqlite	(StrOpt) File name of clean sqlite db
sqlite_db=nova.sqlite	(StrOpt) the filename to use with sqlite
sqlite_synchronous=true	(BoolOpt) If passed, use synchronous mode for sqlite
sql_max_pool_size=5	(IntOpt) Maximum number of SQL connections to keep open in a pool. If set, this value is used for pool_size within sqlalchemy.
sql_min_pool_size=1	(IntOpt) Minimum number of SQL connections to keep open in a pool.
sql_max_overflow=None	(IntOpt) If set, this value is used as max_overflow within sqlalchemy.
dbapi_use_tpool=False	(BoolOpt) Enable the experimental use of thread pooling for all DB API calls.

## Configuring the Oslo RPC Messaging System

OpenStack projects use an open standard for messaging middleware known as AMQP. This messaging middleware enables the OpenStack services which will exist across multiple servers to talk to each other. OpenStack Oslo RPC supports two implementations of AMQP: RabbitMQ and Qpid.

### Configuration for RabbitMQ

OpenStack Oslo RPC uses RabbitMQ by default. This section discusses the configuration options that are relevant when RabbitMQ is used. The `rpc_backend` option is not required as long as RabbitMQ is the default messaging system. However, if it is included the configuration, it must be set to `nova.rpc.impl_kombu`.

```
rpc_backend=nova.rpc.impl_kombu
```

The following tables describe the rest of the options that can be used when RabbitMQ is used as the messaging system. You can configure the messaging communication for different installation scenarios as well as tune RabbitMQ's retries and the size of the RPC thread pool. If you want to monitor notifications through RabbitMQ, you must set the `notification_driver` option in `nova.conf` to

`nova.notifier.rabbit_notifier`. The default for sending usage data is 60 seconds plus a randomized 0-60 seconds.

**Table 4.10. Description of `nova.conf` configuration options for Remote Procedure Calls and RabbitMQ Messaging**

Configuration option	Default	Description
rabbit_host	localhost	IP address; Location of RabbitMQ installation.
rabbit_password	guest	String value; Password for the RabbitMQ server.
rabbit_port	5672	Integer value; Port where RabbitMQ server is running/listening.
rabbit_userid	guest	String value; User ID used for RabbitMQ connections.
rabbit_virtual_host	/	Location of a virtual RabbitMQ installation.

**Table 4.11. Description of `nova.conf` configuration options for Tuning RabbitMQ Messaging**

Configuration option	Default	Description
--rabbit_max_retries	0	Integer value; maximum retries with trying to connect to RabbitMQ(the default of 0 implies an infinite retry count).
rabbit_retry_interval	1	Integer value: RabbitMQ connection retry interval.
rpc_thread_pool_size	1024	Integer value: Size of Remote Procedure Call thread pool.

## Configuration for Qpid

This section discusses the configuration options that are relevant if Qpid is used as the messaging system for OpenStack Oslo RPC. Qpid is not the default messaging system, so it must be enabled by setting the `rpc_backend` option in `nova.conf`.

```
rpc_backend=nova.rpc.impl_qpid
```

This next critical option points the compute nodes to the Qpid broker (server). Set `qpid_hostname` in `nova.conf` to be the hostname where the broker is running.



### Note

The `--qpid_hostname` option accepts a value in the form of either a hostname or an IP address.

```
qpid_hostname=hostname.example.com
```

If the Qpid broker is listening on a port other than the AMQP default of 5672, you will need to set the `qpid_port` option:

```
qpid_port=12345
```

If you configure the Qpid broker to require authentication, you will need to add a username and password to the configuration:

```
qpid_username=username
qpid_password=password
```

By default, TCP is used as the transport. If you would like to enable SSL, set the `qpid_protocol` option:

```
qpid_protocol=ssl
```

The following table lists the rest of the options used by the Qpid messaging driver for OpenStack Oslo RPC. It is not common that these options are used.

**Table 4.12. Remaining `nova.conf` configuration options for Qpid support**

Configuration option	Default	Description
<code>qpid_sasl_mechanisms</code>	(Qpid default)	String value: A space separated list of acceptable SASL mechanisms to use for authentication.
<code>qpid_reconnect_timeout</code>	(Qpid default)	Integer value: The number of seconds to wait before deciding that a reconnect attempt has failed.
<code>qpid_reconnect_limit</code>	(Qpid default)	Integer value: The limit for the number of times to reconnect before considering the connection to be failed.
<code>qpid_reconnect_interval_min</code>	(Qpid default)	Integer value: Minimum number of seconds between connection attempts.
<code>qpid_reconnect_interval_max</code>	(Qpid default)	Integer value: Maximum number of seconds between connection attempts.
<code>qpid_reconnect_interval</code>	(Qpid default)	Integer value: Equivalent to setting <code>qpid_reconnect_interval_min</code> and <code>qpid_reconnect_interval_max</code> to the same value.
<code>qpid_heartbeat</code>	5	Integer value: Seconds between heartbeat messages sent to ensure that the connection is still alive.
<code>qpid_tcp_nodelay</code>	True	Boolean value: Disable the Nagle algorithm.

## Common Configuration for Messaging

This section lists options that are common between both the RabbitMQ and Qpid messaging drivers.

**Table 4.13. Description of nova.conf configuration options for Customizing Exchange or Topic Names**

Configuration option	Default	Description
control_exchange	nova	String value; Name of the main AMQP exchange to connect to if using RabbitMQ or Qpid for RPC (not Zeromq).
ajax_console_proxy_topic	ajax_proxy	String value; Topic that the ajax proxy nodes listen on
console_topic	console	String value; The topic console proxy nodes listen on
network_topic	network	String value; The topic network nodes listen on.
scheduler_topic	scheduler	String value; The topic scheduler nodes listen on.
volume_topic	volume	String value; Name of the topic that volume nodes listen on

## Configuring the Compute API

### Configuring Compute API password handling

The OpenStack Compute API allows the user to specify an admin password when creating (or rebuilding) a server instance. If no password is specified, a randomly generated password is used. The password is returned in the API response.

In practice, the handling of the admin password depends on the hypervisor in use, and may require additional configuration of the instance, such as installing an agent to handle the password setting. If the hypervisor and instance configuration do not support the setting of a password at server create time, then the password returned by the create API call will be misleading, since it was ignored.

To prevent this confusion, the configuration option `enable_instance_password` can be used to disable the return of the admin password for installations that don't support setting instance passwords.

**Table 4.14. Description of nova.conf API related configuration options**

Configuration option	Default	Description
<code>enable_instance_password</code>	true	When true, the create and rebuild compute API calls return the server admin password. When false, the server admin password is not included in API responses.

## Configuring Compute API Rate Limiting

OpenStack Compute supports API rate limiting for the OpenStack API. The rate limiting allows an administrator to configure limits on the type and number of API calls that can be made in a specific time interval.

When API rate limits are exceeded, HTTP requests will return an error with a status code of 413 "Request entity too large", and will also include a 'Retry-After' HTTP header. The response body will include the error details, and the delay before the request should be retried.

Rate limiting is not available for the EC2 API.

## Specifying Limits

Limits are specified using five values:

- The **HTTP method** used in the API call, typically one of GET, PUT, POST, or DELETE.
- A **human readable URI** that is used as a friendly description of where the limit is applied.
- A **regular expression**. The limit will be applied to all URI's that match the regular expression and HTTP Method.
- A **limit value** that specifies the maximum count of units before the limit takes effect.
- An **interval** that specifies time frame the limit is applied to. The interval can be SECOND, MINUTE, HOUR, or DAY.

Rate limits are applied in order, relative to the HTTP method, going from least to most specific. For example, although the default threshold for POST to \*/servers is 50 per day, one cannot POST to \*/servers more than 10 times within a single minute because the rate limits for any POST is 10/min.

## Default Limits

OpenStack Compute is normally installed with the following limits enabled:

**Table 4.15. Default API Rate Limits**

HTTP method	API URI	API regular expression	Limit
POST	any URI (*)	.*	10 per minute
POST	/servers	^/servers	50 per day
PUT	any URI (*)	.*	10 per minute
GET	*changes-since*	.*changes-since.*	3 per minute
DELETE	any URI (*)	.*	100 per minute

## Configuring and Changing Limits

The actual limits are specified in the file `etc/nova/api-paste.ini`, as part of the WSGI pipeline.

To enable limits, ensure the 'ratelimit' filter is included in the API pipeline specification. If the 'ratelimit' filter is removed from the pipeline, limiting will be disabled. There should also be a definition for the ratelimit filter. The lines will appear as follows:

```
[pipeline:openstack_compute_api_v2]
pipeline = faultwrap authtoken keystonecontext ratelimit osapi_compute_app_v2

[pipeline:openstack_volume_api_v1]
pipeline = faultwrap authtoken keystonecontext ratelimit osapi_volume_app_v1

[filter:ratelimit]
paste.filter_factory = nova.api.openstack.compute.
limits:RateLimitingMiddleware.factory
```

To modify the limits, add a 'limits' specification to the [filter:ratelimit] section of the file. The limits are specified in the order HTTP method, friendly URI, regex, limit, and interval. The following example specifies the default rate limiting values:

```
[filter:ratelimit]
paste.filter_factory = nova.api.openstack.compute.
limits:RateLimitingMiddleware.factory
limits =(POST, "*", .*, 10, MINUTE);(POST, "*/servers", ^/servers, 50, DAY);
(PUT, "*", .*, 10, MINUTE);(GET, "*changes-since*", .*changes-since.* , 3,
MINUTE);(DELETE, "*", .*, 100, MINUTE)
```

## Configuring the EC2 API

You can use `nova.conf` configuration options to control which network address and port the EC2 API will listen on, the formatting of some API responses, and authentication related options.

To customize these options for OpenStack EC2 API, use these configuration option settings.

**Table 4.16. Description of `nova.conf` file configuration options for EC2 API**

Configuration option=Default value	(Type) Description
<code>ec2_listen=0.0.0.0</code>	(StrOpt) IP address for EC2 API to listen
<code>ec2_listen_port=8773</code>	(IntOpt) port for ec2 api to listen
<code>ec2_private_dns_show_ip=false</code>	(BoolOpt) Return the IP address as private dns hostname in describe instances, else returns instance name
<code>keystone_ec2_url=http://localhost:5000/v2.0/ec2tokens</code>	(StrOpt) URL to get token from ec2 request
<code>lockout_attempts=5</code>	(IntOpt) Number of failed auths before lockout.
<code>lockout_minutes=15</code>	(IntOpt) Number of minutes to lockout if triggered.
<code>lockout_window=15</code>	(IntOpt) Number of minutes for lockout window.

## Configuring Quotas

For tenants, quota controls are available to limit the (flag and default shown in parenthesis):

- Number of volumes which may be created (`volumes=10`)
- Total size of all volumes within a project as measured in GB (`gigabytes=1000`)

- Number of instances which may be launched (`instances=10`)
- Number of processor cores which may be allocated (`cores=20`)
- Publicly accessible IP addresses for persistence in DNS assignment (`floating_ips=10`)
- Privately (or publicly) accessible IP addresses for management purposes (`fixed_ips=-1` unlimited)
- Amount of RAM that can be allocated in MB (`ram=512000`)
- Number of files that can be injected (`injected_files=5`)
- Maximal size of injected files in B (`injected_file_content_bytes=10240`)
- Number of security groups that may be created (`security_groups=10`)
- Number of rules per security group (`security_group_rules=20`)

The defaults may be modified by setting the variable in `nova.conf`, then restarting the `nova-api` service.

To modify a value for a specific project, the **nova-manage** command should be used. For example:

```
$ nova-manage project quota --project=1113f5f266f3477ac03da4e4f82d0568 --key=cores --value=40
```

Alternately, quota settings are available through the OpenStack Dashboard in the "Edit Project" page.

# 5. Configuration: nova.conf

## Table of Contents

File format for nova.conf .....	60
List of configuration options .....	62

## File format for nova.conf

### Overview

The Compute service supports a large number of configuration options. These options are specified in a configuration file whose default location in /etc/nova/nova.conf.

The configuration file is in [INI file format](#), with options specified as key=value pairs, grouped into sections. Almost all of the configuration options are in the DEFAULT section. Here's a brief example:

```
[DEFAULT]
debug=true
verbose=true

[trusted_computing]
server=10.3.4.2
```

## Types of configuration options

Each configuration option has an associated type that indicates what values can be set. The supported option types are as follows:

BoolOpt      Boolean option. Value must be either true or false. Example:

```
debug=false
```

StrOpt      String option. Value is an arbitrary string. Example:

```
my_ip=10.0.0.1
```

IntOption      Integer option. Value must be an integer. Example:

```
glance_port=9292
```

MultiStrOpt      String option. Same as StrOpt, except that it can be declared multiple times to indicate multiple values. Example:

```
ldap_dns_servers=dns1.example.org
ldap_dns_servers=dns2.example.org
```

ListOpt      List option. Value is a list of arbitrary strings separated by commas. Example:

```
enabled_apis=ec2,osapi_compute,metadata
```

FloatOpt      Floating-point option. Value must be a floating-point number. Example:

```
ram_allocation_ratio=1.5
```



## Important

Nova options should *not* be quoted.

## Sections

Configuration options are grouped by section. The Compute config file supports the following sections.

[DEFAULT]	Almost all of the configuration options are organized into this section. If the documentation for a configuration option does not specify its section, assume that it should be placed in this one.
[cells]	The <code>cells</code> section is used for options for configuring cells functionality. See the <a href="#">Cells</a> section of the OpenStack Compute Admin Manual for more details.
[baremetal]	This section is used for options that relate to the baremetal hypervisor driver.
[conductor]	The <code>conductor</code> section is used for options for configuring the nova-conductor service.
[trusted_computing]	The <code>trusted_computing</code> section is used for options that relate to the trusted computing pools functionality. Options in this section describe how to connect to a remote attestation service.

## Variable substitution

The configuration file supports variable substitution. Once a configuration option is set, it can be referenced in later configuration values when preceded by \$. Consider the following example where `my_ip` is defined and then `$my_ip` is used as a variable.

```
my_ip=10.2.3.4
glance_host=$my_ip
metadata_host=$my_ip
```

If you need a value to contain the \$ symbol, escape it by doing \$\$ . For example, if your LDAP DNS password was \$xkj432, you would do:

```
ldap_dns_password=$$xkj432
```

The Compute code uses Python's `string.Template.safe_substitute()` method to implement variable substitution. For more details on how variable substitution is resolved, see [Python documentation on template strings](#) and [PEP 292](#).

## Whitespace

To include whitespace in a configuration value, use a quoted string. For example:

```
ldap_dns_password='a password with spaces'
```

## Specifying an alternate location for nova.conf

The configuration file is loaded by all of the nova-\* services, as well as the **nova-manage** command-line tool. To specify an alternate location for the configuration file, pass the `--config-file /path/to/nova.conf` argument when starting a nova-\* service or calling **nova-manage**.

## List of configuration options

For a complete list of all available configuration options for each OpenStack Compute service, run `bin/nova-<servicename> -help`.

**Table 5.1. Description of common `nova.conf` configuration options for the Compute API, RabbitMQ, EC2 API, S3 API, instance types**

Configuration option=Default value	(Type) Description
allow_resize_to_same_host=false	(BoolOpt) Allow destination machine to match source for resize. Useful when testing in single-host environments. If you have separate configuration files for separate services, this flag is required on both nova-api and nova-compute.
api_paste_config=api-paste.ini	(StrOpt) File name for the paste.deploy config for nova-api
api_rate_limit=true	(BoolOpt) whether to rate limit the Compute API
api_url=	(StrOpt) URL for the Zone's Auth API
auth_blob=	(StrOpt) To be written, found in /nova/scheduler/filters/trusted_filter.py, related to FLAGS.trusted_computing.auth_blob.
aws_access_key_id=admin	(StrOpt) AWS Access ID
aws_secret_access_key=admin	(StrOpt) AWS Access Key
backdoor_port=<None>	(IntOpt) Port for eventlet backdoor to listen
bandwidth_poll_interval=600	(IntOpt) Interval to pull bandwidth usage info
bindir=\$pybasedir/bin	(StrOpt) Directory where nova binaries are installed
cache_images=true	(BoolOpt) Cache glance images locally
cert_manager=nova.cert.manager.CertManager	(StrOpt) full class name for the Manager for cert
cert_topic=cert	(StrOpt) the topic cert nodes listen on
claim_timeout_seconds=600	(IntOpt) Found in /nova/compute/resource_tracker.py
compute_api_class=nova.compute.api.API	(StrOpt) The full class name of the Compute API class to use
compute_manager=nova.compute.manager.ComputeManager	(StrOpt) full class name for the Manager for compute
compute_topic=compute	(StrOpt) the topic compute nodes listen on
config_file=/etc/nova/nova.conf	(MultiStrOpt) Path to a config file to use. Multiple config files can be specified, with values in later files taking precedence. The default files used are: []
compute_driver='nova.virt.connection.get_connection'	String value; Driver to use for controlling virtualization. For convenience if the driver exists under the nova.virt namespace, nova.virt can be removed. There are 5 drivers in core openstack: fake.FakeDriver, libvirt.LibvirtDriver, baremetal.BareMetalDriver, xenapi.XenAPIDriver, vmwareapi.VMwareESXDriver, vmwareapi.VMwareVCDriver.
console_manager=nova.console.manager.ConsoleProxyManager	(StrOpt) full class name for the Manager for console proxy
console_topic=console	(StrOpt) the topic console proxy nodes listen on

Configuration option=Default value	(Type) Description
control_exchange=nova	(StrOpt) AMQP exchange to connect to if using RabbitMQ or Qpid for RPC (not Zeromq).  Currently you cannot set different exchange values for volumes and networks, for example.
debug=false	(BoolOpt) Print debugging output
default_access_ip_network_name=<None>	(StrOpt) Name of network to use to set access ips for instances
default_ephemeral_format=<None>	(StrOpt) The default format an ephemeral_volume will be formatted with on creation.
default_image=ami-11111	(StrOpt) default image to use, testing only
default_instance_type=m1.small	(StrOpt) default instance type to use, testing only
default_project=openstack	(StrOpt) the default project to use for OpenStack
default_schedule_zone=<None>	(StrOpt) availability zone to use when user doesn't specify one
default_scheduler_driver=<None>	(StrOpt)
disable_process_locking=false	(BoolOpt) Whether to disable inter-process locks
ec2_dmz_host=\$my_ip	(StrOpt) the internal IP address of the EC2 API server
ec2_host=\$my_ip	(StrOpt) the IP of the ec2 api server
ec2_path=/services/Cloud	(StrOpt) the path prefix used to call the EC2 API server
ec2_port=8773	(IntOpt) the port of the EC2 API server
ec2_scheme=http	(StrOpt) the protocol to use when connecting to the EC2 API server (http, https)
ec2_strict_validation=true	(BoolOpt) Enables strict validation for EC2 API server requests
ec2_workers=<None>	(StrOpt) To be written; Found in /nova/service.py
enable_instance_password=true	(BoolOpt) When true, Compute creates a random password for the instance at create time. Users can get the password from the return value of API call for the instance creation (or through their Dashboard if the Dashboard returns the password visibly). Note that the password isn't stored anywhere, it is returned only once.
enabled_apis=ec2,osapi_compute,osapi_volume,metadata	(ListOpt) a list of APIs to enable by default
fake_network=false	(BoolOpt) If passed, use fake network devices and addresses
fake_rabbit=false	(BoolOpt) If passed, use a fake RabbitMQ provider
fatal_deprecations=false	(BoolOpt) To be written; Found in /nova/common/deprecated.py
firewall_driver=nova.virt.firewall.libvirt.IptablesFirewallDriver	(StrOpt) Firewall driver (defaults to iptables)
floating_ip_dns_manager=nova.network.dns_driver.DNSDriver	(StrOpt) full class name for the DNS Manager for floating IPs
glance_api_insecure=false	(BoolOpt) Allow to perform insecure SSL (https) requests to glance
glance_api_servers=\$glance_host:\$glance_port	(ListOpt) A list of the glance API servers available to nova ([hostname ip]:port)
glance_host=\$my_ip	(StrOpt) default glance hostname or IP
glance_num_retries=0	(IntOpt) Number retries when downloading an image from glance
glance_port=9292	(IntOpt) default glance port
host=MGG2WEDRJM	(StrOpt) Name of this node. This can be an opaque identifier. It is not necessarily a hostname, FQDN, or IP address.

Configuration option=Default value	(Type) Description
image_info_filename_pattern=\$instances_path/	(StrOpt) Used for image caching; found in /nova/virt/libvirt/utils.py
image_service=nova.image.glance.GlanceImageService	(StrOpt) The service to use for retrieving and searching images.
instance_build_timeout=0	(StrOpt) To be written; found in /nova/compute/manager.py
instance_dns_domain=	(StrOpt) full class name for the DNS Zone for instance IPs
instance_dns_manager=nova.network.dns_driver.DNSDriver	(StrOpt) full class name for the DNS Manager for instance IPs
instance_usage_audit_period=month	(StrOpt) time period to generate instance usages for. Time period must be hour, day, month or year
instance_uuid_format= [instance: %(uuid)s]	(StrOpt) To be written; found in /nova/openstack/common/log.py
iptables_bottom_regex=	(StrOpt) Regular expressions to match iptables rules that should always be on the bottom.
iptables_top_regex=	(StrOpt) Regular expressions to match iptables rules that should always be on the top.
isolated_hosts=	(ListOpt) Host reserved for specific images
isolated_images=	(ListOpt) Images to run on isolated host
lock_path=\$pybasedir	(StrOpt) Directory to use for lock files
log_config=<None>	(StrOpt) 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.
log_date_format=%Y-%m-%d %H:%M:%S	(StrOpt) Format string for %(asctime)s in log records. Default: %default
log_dir=<None>	(StrOpt) (Optional) The directory to keep log files in (will be prepended to -logfile)
log_file=<None>	(StrOpt) (Optional) Name of log file to output to. If not set, logging will go to stdout.
log_format= "%(asctime)s %(levelname)8s [%(name)s] %(message)s"	(StrOpt) A logging.Formatter log message format string which may use any of the available logging.LogRecord attributes. Default: %default
logdir=<None>	(StrOpt) Log output to a per-service log file in named directory
logfile=<None>	(StrOpt) Log output to a named file
logfile_mode=0644	(StrOpt) Default file mode used when creating log files
memcached_servers=<None>	(ListOpt) Memcached servers or None for in process cache.
metadata_host=\$my_ip	(StrOpt) the IP address for the metadata API server
metadata_port=8775	(IntOpt) the port for the metadata API port
monkey_patch=false	(BoolOpt) Whether to log monkey patching
monkey_patch_modules=nova.api.ec2.cloud:nova.notifier.api:nova.compute.api:nova.notifier.api.notify_decorator	(ListOpt) list of modules/decorators to monkey patch
multi_instance_display_name_template=%s(name)s-%(uuid)s	(StrOpt) When creating multiple instances with a single request using the os-multiple-create API extension, this template will be used to build the display name for each instance. The benefit is that the instances end up with different hostnames. To maintain the legacy behaviour of every instance having the same name, set this option to "%(name)s". Valid keys for the template are: name, uuid, count.

Configuration option=Default value	(Type) Description
my_ip=192.168.1.82	(StrOpt) IP address of this host; change my_ip to match each host when copying nova.conf files to multiple hosts.
network_api_class=nova.network.api.API	(StrOpt) The full class name of the network API class to use
network_driver=nova.network.linux_net	(StrOpt) Driver to use for network creation
network_manager=nova.network.manager.VlanManager	(StrOpt) Full class name for the Manager for network
network_topic=network	(StrOpt) The topic network nodes listen on
node_availability_zone=nova	(StrOpt) Availability zone of this node
non_inheritable_image_properties=['cache_in_nova', 'instance_uuid', 'user_id', 'image_type', 'backup_type', 'min_ram', 'min_disk']	(ListOpt) These are image properties which a snapshot should not inherit from an instance
notification_driver=nova.notifier.no_op_notifier	(StrOpt) Default driver for sending notifications for RabbitMQ. By default, set to not send entries to the notifications.info queue. Set to nova.notifier.rabbit_notifier to send notifications.
null_kernel=nokernel	(StrOpt) kernel image that indicates not to use a kernel, but to use a raw disk image instead
osapi_compute_ext_list=	(ListOpt) Specify list of extensions to load when using osapi_compute_extension option with nova.api.openstack.compute.contrib.select_extensions
osapi_compute_extension=nova.api.openstack.compute.contrib.select_extensions	(MultiStrOpt) Extensions to load
osapi_compute_link_prefix=<None>	(StrOpt) Base URL that will be presented to users in links to the OpenStack Compute API
osapi_glance_link_prefix=<None>	(StrOpt) Base URL that will be presented to users in links to glance resources
osapi_max_limit=1000	(IntOpt) the maximum number of items returned in a single response from a collection resource
osapi_path=/v1.1/	(StrOpt) the path prefix used to call the OpenStack Compute API server
osapi_scheme=http	(StrOpt) the protocol to use when connecting to the OpenStack Compute API server (http, https)
osapi_volume_ext_list=	(ListOpt) Specify list of extensions to load when using osapi_volume_extension option with nova.api.openstack.volume.contrib.select_extensions
osapi_volume_extension=nova.api.openstack.volume.contrib.select_extensions	(MultiStrOpt) Extensions to load
password_length=12	(IntOpt) Length of generated instance admin passwords
pybasedir=/usr/lib/python/site-packages	(StrOpt) Directory where the nova python module is installed
rabbit_durable_queues=false	(BoolOpt) use durable queues in RabbitMQ
rabbit_host=localhost	(StrOpt) the RabbitMQ host
rabbit_max_retries=0	(IntOpt) maximum retries with trying to connect to RabbitMQ (the default of 0 implies an infinite retry count)
rabbit_password=guest	(StrOpt) the RabbitMQ password
rabbit_port=5672	(IntOpt) the RabbitMQ port
rabbit_retry_backoff=2	(IntOpt) how long to backoff for between retries when connecting to RabbitMQ
rabbit_retry_interval=1	(IntOpt) how frequently to retry connecting with RabbitMQ
rabbit_use_ssl=false	(BoolOpt) connect over SSL for RabbitMQ
rabbit_userid=guest	(StrOpt) the RabbitMQ userid

Configuration option=Default value	(Type) Description
rabbit_virtual_host=/	(StrOpt) the RabbitMQ virtual host
reclaim_instance_interval=0	(IntOpt) Interval in seconds for reclaiming deleted instances
region_list=	(ListOpt) list of region=fqdn pairs separated by commas
resume_guests_state_on_host_boot=false	(BoolOpt) Whether to start guests that were running before the host rebooted. If enabled, this option causes guests assigned to the host to be restarted when nova-compute starts, <i>if</i> they had been active on the host while nova-compute last ran. If such a guest is already found to be running, it is left untouched.
rootwrap_config=sudo nova-rootwrap /etc/nova/rootwrap.conf	(StrOpt) Command prefix to use for running commands as root. Note that the configuration file (and executable) used here must match the one defined in the sudoers entry from packagers, otherwise the commands are rejected.
s3_dmz=\$my_ip	(StrOpt) hostname or IP for the instances to use when accessing the S3 API
s3_host=\$my_ip	(StrOpt) hostname or IP for OpenStack to use when accessing the S3 API
s3_port=3333	(IntOpt) port used when accessing the S3 API
scheduler_manager=nova.scheduler.manager.SchedulerManager	(StrOpt) full class name for the Manager for scheduler
scheduler_topic=scheduler	(StrOpt) the topic scheduler nodes listen on
security_group_api=nova	(StrOpt) If using nova security groups set to nova. If set to quantum all nova security group api requests will be proxied to quantum to handle.
service_down_time=60	(IntOpt) maximum time since last check-in for up service
state_path=\$pybasedir	(StrOpt) Top-level directory for maintaining nova's state
stub_network=False	(StrOpt) Stub network related code
syslog-log-facility=LOG_USER	(StrOpt) syslog facility to receive log lines
tempdir=<None>	(StrOpt) Although the temporary directory used can be controlled via environment variables, this patch provides a way to define it explicitly via a config option. The default value is None, which behaves per <a href="#">this documentation</a> .
use_cow_images=true	(BoolOpt) Whether to use cow images
use_stderr=true	(BoolOpt) Log output to standard error
use-syslog=false	(BoolOpt) Use syslog for logging.
verbose=false	(BoolOpt) Print more verbose output
volume_api_class=nova.volume.api.API	(StrOpt) The full class name of the volume API class to use
volume_manager=nova.volume.manager.VolumeManager	(StrOpt) full class name for the Manager for volume
volume_topic=volume	(StrOpt) the topic volume nodes listen on
vpn_image_id=0	(StrOpt) image id used when starting up a cloupipe VPN server
vpn_key_suffix=-vpn	(StrOpt) Suffix to add to project name for vpn key and secgroups
zombie_instance_updated_at_window=172800	(IntOpt) Number of seconds zombie instances are cleaned up.

**Table 5.2. Description of nova.conf configuration options for databases**

Configuration option=Default value	(Type) Description
db_backend=sqlalchemy	(StrOpt) The backend to use for db
db_driver=nova.db	(StrOpt) driver to use for database access

Configuration option=Default value	(Type) Description
sql_connection=sqlite:///\$state_path/\$sqlite_db	(StrOpt) The SQLAlchemy connection string used to connect to the database
sql_connection_debug=0	(IntOpt) Verbosity of SQL debugging information. 0=None, 100=Everything
sql_connection_trace=false	(BoolOpt) Add python stack traces to SQL as comment strings
sql_idle_timeout=3600	(IntOpt) timeout before idle sql connections are reaped
sql_max_retries=10	(IntOpt) maximum db connection retries during startup. (setting -1 implies an infinite retry count)
sql_retry_interval=10	(IntOpt) interval between retries of opening a sql connection
sqlite_clean_db=clean.sqlite	(StrOpt) File name of clean sqlite db
sqlite_db=nova.sqlite	(StrOpt) the filename to use with sqlite
sqlite_synchronous=true	(BoolOpt) If passed, use synchronous mode for sqlite
sql_max_pool_size=5	(IntOpt) Maximum number of SQL connections to keep open in a pool. If set, this value is used for pool_size within sqlalchemy.
sql_min_pool_size=1	(IntOpt) Minimum number of SQL connections to keep open in a pool.
sql_max_overflow=None	(IntOpt) If set, this value is used as max_overflow within sqlalchemy.
dbapi_use_tpool=False	(BoolOpt) Enable the experimental use of thread pooling for all DB API calls.

**Table 5.3. Description of nova.conf configuration options for IPv6**

Configuration option=Default value	(Type) Description
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

**Table 5.4. Description of nova.conf log file configuration options**

Configuration option=Default value	(Type) Description
default_log_levels="amqplib=WARN,sqlalchemy=WARN,boto=WARN,suds=INFO,eventlet.wsgi.server=WARN"	(ListOpt) list of logger=LEVEL pairs
instance_format=[instance: %(uuid)s]	(StrOpt) If an instance is passed with the log message, format it like this
instance_uuid_format=[instance: %(uuid)s]	(StrOpt) If an instance UUID is passed with the log message, format it like this
log_config=<None>	(StrOpt) 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.
log_date_format=%Y-%m-%d %H:%M:%S	(StrOpt) Format string for %(asctime)s in log records. Default: %default
log_dir=<None>	(StrOpt) (Optional) The directory to keep log files in (will be prepended to -logfile)
log_file=<None>	(StrOpt) (Optional) Name of log file to output to. If not set, logging will go to stdout.

Configuration option=Default value	(Type) Description
log_format="%(asctime)s %(levelname)s [%(name)s] %(message)s"	(StrOpt) A logging.Formatter log message format string which may use any of the available logging.LogRecord attributes. Default: %default
logdir=<None>	(StrOpt) Log output to a per-service log file in named directory
logfile=<None>	(StrOpt) Log output to a named file
logfile_mode=0644	(StrOpt) Default file mode used when creating log files
logging_context_format_string="%(asctime)s %(levelname)s [%(name)s] [%{request_id}s %{user_id}s %{project_id}s] %(instance)s%(message)s"	(StrOpt) format string to use for log messages with context
logging_debug_format_suffix="from (pid=%(process)d) %(funcName)s %(pathname)s:%(lineno)d"	(StrOpt) data to append to log format when level is DEBUG
logging_default_format_string="%(asctime)s %(levelname)s %(name)s [-] %(instance)s%(message)s"	(StrOpt) format string to use for log messages without context
logging_exception_prefix="%(asctime)s TRACE %(name)s %(instance)s"	(StrOpt) prefix each line of exception output with this format
publish_errors=false	(BoolOpt) publish error events
publish_errors=false	(BoolOpt) publish error events
use_syslog=false	(BoolOpt) Use syslog for logging
syslog_log_facility=LOG_USER	(StrOpt) syslog facility to receive log lines

**Table 5.5. Description of nova.conf file configuration options for nova-services**

Configuration option=Default value	(Type) Description
enable_new_services=true	(BoolOpt) Services to be added to the available pool on create
instance_name_template=instance-%08x	(StrOpt) Template string to be used to generate instance names
matchmaker_ringfile=/etc/nova/matchmaker_ring.json	(StrOpt) When using rpc_backend set to ZeroMQ (nova.rpc.impl_zmq), enables use of a static hash table from a JSON file, cycles hosts per bare topic to create a directed topic.
metadata_listen=0.0.0.0	(StrOpt) IP address for metadata api to listen
metadata_listen_port=8775	(IntOpt) port for metadata api to listen
metadata_manager=nova.api.manager.MetadataManager	(StrOpt) OpenStack metadata service manager
osapi_compute_listen=0.0.0.0	(StrOpt) IP address for OpenStack API to listen
osapi_compute_listen_port=8774	(IntOpt) list port for osapi compute
osapi_compute_unique_server_name_scope=""	(StrOpt) When set, the Compute API will consider duplicate hostnames (case insensitive) invalid within the specified scope. Valid scope settings are empty, "project" or "global".
osapi_volume_listen=0.0.0.0	(StrOpt) IP address for OpenStack Volume API to listen
osapi_volume_listen_port=8776	(IntOpt) port for os volume api to listen
periodic_fuzzy_delay=60	(IntOpt) range of seconds to randomly delay when starting the periodic task scheduler to reduce stampeding. (Disable by setting to 0)
periodic_interval=60	(IntOpt) seconds between running periodic tasks
report_interval=10	(IntOpt) seconds between nodes reporting state to datastore
rpc_backend=nova.rpc.impl_kombu	(StrOpt) The messaging module to use, defaults to kombu.

Configuration option=Default value	(Type) Description
servicegroup_driver=db	(StrOpt) The driver for servicegroup service which maintains heartbeat information of Nova services/nodes is by default 'db' (Database). Could be 'mc' for using Memcached instead (more lightweight and better for large scale deployments).
snapshot_name_template=snapshot-%08x	(StrOpt) Template string to be used to generate snapshot names
volume_name_template=volume-%s	(StrOpt) Template string to be used to generate instance names

**Table 5.6. Description of nova.conf file configuration options for credentials (crypto)**

Configuration option=Default value	(Type) Description
ca_file=cacert.pem	(StrOpt) Filename of root CA (Certificate Authority)
ca_path=\$state_path/CA	(StrOpt) Where we keep our root CA
crl_file=crl.pem	(StrOpt) Filename of root Certificate Revocation List
key_file=private/cakey.pem	(StrOpt) Filename of private key
keys_path=\$state_path/keys	(StrOpt) Where we keep our keys
project_cert_subject="/C=US/ST=California/O=OpenStack/OU=NovaDev/CN=project-ca%.16s-%s"	(StrOpt) Subject for certificate for projects, %s for project, timestamp
use_project_ca=false	(BoolOpt) Whether to use a CA for each project (tenant)
user_cert_subject="/C=US/ST=California/O=OpenStack/OU=NovaDev/CN=%16s-%16s-%s"	(StrOpt) Subject for certificate for users, %s for project, user, timestamp

**Table 5.7. Description of nova.conf file configuration options for policies (policy.json)**

Configuration option=Default value	(Type) Description
policy_default_rule=default	(StrOpt) Rule checked when requested rule is not found
policy_file=policy.json	(StrOpt) JSON file representing policy
allow_instance_snapshots=true	(BoolOpt) Permit instance snapshot operations.
osapi_max_request_body_size=114688	(BoolOpt)

**Table 5.8. Description of nova.conf file configuration options for quotas**

Configuration option=Default value	(Type) Description
max_age=0	(IntOpt) number of seconds between subsequent usage refreshes
quota_cores=20	(IntOpt) number of instance cores allowed per project (tenant)
quota_driver=nova.quota.DbQuotaDriver	(StrOpt) Default driver to use for quota checks
quota_floating_ips=10	(IntOpt) number of floating ips allowed per project (tenant)
quota_fixed_ips=-1	(IntOpt) number of fixed ips allowed per project (this should be at least the number of instances allowed.) -1 is unlimited.
quota_gigabytes=1000	(IntOpt) number of volume gigabytes allowed per project (tenant)
quota_injected_file_content_bytes=10240	(IntOpt) number of bytes allowed per injected file
quota_injected_file_path_bytes=255	(IntOpt) number of bytes allowed per injected file path

Configuration option=Default value	(Type) Description
quota_injected_files=5	(IntOpt) number of injected files allowed
quota_instances=10	(IntOpt) number of instances allowed per project (tenant)
quota_key_pairs=100	(IntOpt) number of key pairs allowed per user
quota_metadata_items=128	(IntOpt) number of metadata items allowed per instance
quota_ram=51200	(IntOpt) megabytes of instance ram allowed per project (tenant)
quota_security_group_rules=20	(IntOpt) number of security rules per security group
quota_security_groups=10	(IntOpt) number of security groups per project (tenant)
quota_volumes=10	(IntOpt) number of volumes allowed per project (tenant)
reservation_expire=86400	(IntOpt) number of seconds until a reservation expires
until_refresh=0	(IntOpt) count of reservations until usage is refreshed

**Table 5.9. Description of nova.conf file configuration options for testing purposes**

Configuration option=Default value	(Type) Description
allowed_rpc_exception_modules=['nova.exception']	(IntOpt) Modules of exceptions that are permitted to be recreated upon receiving exception data from an rpc call
consoleauth_topic=consoleauth	(StrOpt) the topic console auth proxy nodes listen on
fake_tests=true	(BoolOpt) should we use everything for testing
find_host_timeout=30	(StrOpt) Timeout after NN seconds when looking for a host
rpc_conn_pool_size=30	(IntOpt) Size of RPC connection pool
rpc_response_timeout=60	(IntOpt) Seconds to wait for a response from call or multicall
rpc_thread_pool_size=1024	(IntOpt) Size of RPC thread pool
storage_availability_zone=nova	(StrOpt) availability zone of this service
use_local_volumes=true	(BoolOpt) if True, will not discover local volumes
volume_driver=nova.volume.driver.ISCSIDriver	(StrOpt) Driver to use for volume creation
volume_force_update_capabilities=false	(BoolOpt) if True will force update capabilities on each check

**Table 5.10. Description of nova.conf configuration options for authentication**

Configuration option=Default value	(Type) Description
auth_strategy=noauth	(StrOpt) The strategy to use for authentication. Supports noauth or keystone.
auth_token_ttl=3600	(IntOpt) Seconds for auth tokens to linger
ldap_cloudadmin=cn=cloudadmins,ou=Groups,dc=example,dc=com	(StrOpt) cn for Cloud Admins
ldap_developer=cn=developers,ou=Groups,dc=example,dc=com	(StrOpt) cn for Developers
ldap_itsec=cn=itsec,ou=Groups,dc=example,dc=com	(StrOpt) cn for ItSec
ldap_netadmin=cn=netadmins,ou=Groups,dc=example,dc=com	(StrOpt) cn for NetAdmins
ldap_password=changeme	(StrOpt) LDAP password
ldap_project_subtree=ou=Groups,dc=example,dc=com	(StrOpt) OU for Projects
ldap_schema_version=2	(IntOpt) Current version of the LDAP schema
ldap_sysadmin=cn=sysadmins,ou=Groups,dc=example,dc=com	(StrOpt) cn for Sysadmins
ldap_url=ldap://localhost	(StrOpt) Point this at your ldap server
ldap_user_dn=cn=Manager,dc=example,dc=com	(StrOpt) DN of admin user

Configuration option=Default value	(Type) Description
ldap_user_id_attribute=uid	(StrOpt) Attribute to use as id
ldap_user_modify_only=false	(BoolOpt) Modify user attributes instead of creating/deleting
ldap_user_name_attribute=cn	(StrOpt) Attribute to use as name
ldap_user_subtree=ou=Users,dc=example,dc=com	(StrOpt) OU for Users
ldap_user_unit=Users	(StrOpt) OID for Users
role_project_subtree=ou=Groups,dc=example,dc=com	(StrOpt) OU for Roles
allowed_roles=cloudadmin,itsec,sysadmin,netadmin,developer	(ListOpt) Allowed roles for project
auth_driver=nova.auth.dbdriver.DbDriver	(StrOpt) Driver that auth manager uses
credential_cert_file=cert.pem	(StrOpt) Filename of certificate in credentials zip
credential_key_file=pk.pem	(StrOpt) Filename of private key in credentials zip
credential_rc_file=%src	(StrOpt) Filename of rc in credentials zip %s will be replaced by name of the region (nova by default)
credential_vpn_file=nova-vpn.conf	(StrOpt) Filename of certificate in credentials zip
credentials_template=\$pybasedir/nova/auth/novarc.template	(StrOpt) Template for creating users rc file
global_roles=cloudadmin,itsec	(ListOpt) Roles that apply to all projects
superuser_roles=cloudadmin	(ListOpt) Roles that ignore authorization checking completely
vpn_client_template=\$pybasedir/nova/cloudpipe/client.ovpn.template	(StrOpt) Template for creating users VPN file

**Table 5.11. Description of nova.conf file configuration options for LDAP**

Configuration option=Default value	(Type) Description
ldap_cloudadmin= "cn=cloudadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Cloud Admins
ldap_developer= "cn=developers,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Developers
ldap_itsec= "cn=itsec,ou=Groups,dc=example,dc=com"	(StrOpt) CN for ItSec
ldap_netadmin= "cn=netadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for NetAdmins
ldap_password= "changeme"	(StrOpt) LDAP password
ldap_suffix= "cn=example,cn=com"	(StrOpt) LDAP suffix
ldap_use_dumb_member=False	(BoolOpt) Simulates an LDAP member
ldap_project_subtree= "ou=Groups,dc=example,dc=com"	(StrOpt) OU for Projects
ldap_objectClass= inetOrgPerson	(StrOpt) LDAP objectClass to use
ldap_schema_version=2	(IntOpt) Current version of the LDAP schema
ldap_sysadmin= "cn=sysadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Sysadmins
ldap_url= "ldap://localhost"	(StrOpt) Point this at your ldap server
ldap_user= "dc=Manager,dc=example,dc=com"	(StrOpt) LDAP User
ldap_user_tree_dn= "ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
ldap_user_dn= "cn=Manager,dc=example,dc=com"	(StrOpt) DN of Users
ldap_user_objectClass= inetOrgPerson	(StrOpt) DN of Users
ldap_user_id_attribute= cn	(StrOpt) Attribute to use as id
ldap_user_modify_only=false	(BoolOpt) Modify user attributes instead of creating/deleting
ldap_user_name_attribute= cn	(StrOpt) Attribute to use as name

Configuration option=Default value	(Type) Description
ldap_user_subtree= "ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
ldap_user_unit= "Users"	(StrOpt) OID for Users
ldap_tenant_tree_dn="ou=Groups,dc=example,dc=com"	(StrOpt) OU for Tenants
ldap_tenant_objectclass= groupOfNames	(StrOpt) LDAP ObjectClass to use for Tenants
ldap_tenant_id_attribute= cn	(strOpt) Attribute to use as Tenant
ldap_tenant_member_attribute= member	(strOpt) Attribute to use as Member
ldap_role_tree_dn= "ou=Roles,dc=example,dc=com"	(strOpt) OU for Roles
ldap_role_objectclass= organizationalRole	(strOpt) LDAP ObjectClass to use for Roles
ldap_role_project_subtree= "ou=Groups,dc=example,dc=com"	(StrOpt) OU for Roles
ldap_role_member_attribute= roleOccupant	(StrOpt) Attribute to use as Role member
ldap_role_id_attribute= cn	(StrOpt) Attribute to use as Role

**Table 5.12. Description of nova.conf file configuration options for roles and authentication**

Configuration option=Default value	(Type) Description
allowed_roles=cloudadmin,itsec,sysadmin,netadmin,developer	(ListOpt) Allowed roles for project (tenant)
auth_driver=nova.auth.dbdriver.DbDriver	(StrOpt) Driver that auth manager uses
credential_cert_file=cert.pem	(StrOpt) Filename of certificate in credentials zip
credential_key_file=pk.pem	(StrOpt) Filename of private key in credentials zip
credential_rc_file=%src	(StrOpt) Filename of rc in credentials zip %s will be replaced by name of the region (nova by default)
credential_vpn_file=nova-vpn.conf	(StrOpt) Filename of certificate in credentials zip
credentials_template=\$pybasedir/nova/auth/novarc.template	(StrOpt) Template for creating users rc file
global_roles=cloudadmin,itsec	(ListOpt) Roles that apply to all projects (tenants)
superuser_roles=cloudadmin	(ListOpt) Roles that ignore authorization checking completely
vpn_client_template=\$pybasedir/nova/cloudpipe/client.ovpn.template	(StrOpt) Template for creating users vpn file
use_forwarded_for=false	(BoolOpt) Treat X-Forwarded-For as the canonical remote address. Only enable this if you have a sanitizing proxy.

**Table 5.13. Description of nova.conf file configuration options for EC2 API**

Configuration option=Default value	(Type) Description
ec2_listen=0.0.0.0	(StrOpt) IP address for EC2 API to listen
ec2_listen_port=8773	(IntOpt) port for ec2 api to listen
ec2_private_dns_show_ip=false	(BoolOpt) Return the IP address as private dns hostname in describe instances, else returns instance name
keystone_ec2_url=http://localhost:5000/v2.0/ec2tokens	(StrOpt) URL to get token from ec2 request
lockout_attempts=5	(IntOpt) Number of failed auths before lockout.
lockout_minutes=15	(IntOpt) Number of minutes to lockout if triggered.
lockout_window=15	(IntOpt) Number of minutes for lockout window.

**Table 5.14. Description of nova.conf file configuration options for VNC access to guest instances**

Configuration option=Default value	(Type) Description
novncproxy_base_url=http://127.0.0.1:6080/vnc_auto.html	(StrOpt) location of VNC console proxy, in the form "http://127.0.0.1:6080/vnc_auto.html"
vnc_enabled=true	(BoolOpt) enable VNC related features
vnc_keymap=en-us	(StrOpt) keymap for vnc
vnc_require_instance_uuid_as_password=false	(BoolOpt) When set to true, secure VNC connections by requiring the user to enter the instance uuid as the password. This ensures the user is connecting to the correct VNC console.
vncserver_listen=127.0.0.1	(StrOpt) IP address on which instance VNC servers should listen
vncserver_proxyclient_address=127.0.0.1	(StrOpt) the address to which proxy clients (like novavpnproxy) should connect
xvpvncproxy_base_url=http://127.0.0.1:6081/console	(StrOpt) location of nova XCP VNC console proxy, in the form "http://127.0.0.1:6081/console"
xvpvncproxy_host=0.0.0.0	(StrOpt) Address that the XCP VNC proxy should bind to
xvpvncproxy_port=6081	(IntOpt) Port that the XCP VNC proxy should bind to

**Table 5.15. Description of nova.conf [spice] section configuration options for SPICE HTML5 access to guest instances**

Configuration option=Default value	(Type) Description
html5proxy_base_url=http://\$nova-html5proxy_host:6082/spice_auto.html	(StrOpt) location of spice html5 console proxy, in the form "http://\$nova-html5proxy_host:6082/spice_auto.html"
enabled=false	(BoolOpt) enable spice related features
agent_enabled=true	(BoolOpt) enable spice guest agent support
keymap=en-us	(StrOpt) keymap for spice
server_listen=0.0.0.0	(StrOpt) IP address on which instance spice servers should listen
server_proxyclient_address=\$compute_host	(StrOpt) Management IP Address on which instance spicesservers will listen on the compute host.

**Table 5.16. Description of nova.conf file configuration options for networking options**

Configuration option=Default value	(Type) Description
allow_same_net_traffic=true	(BoolOpt) Whether to allow network traffic from same network
defer_iptables_apply=false	(BoolOpt) Whether to batch up the application of IPTables rules during a host restart and apply all at the end of the init phase
dhcp_lease_time=120	(IntOpt) Lifetime of a DHCP lease in seconds
dhcpbridge=\$bindir/nova-dhcpbridge	(StrOpt) location of nova-dhcpbridge
dhcpbridge_flagfile=/etc/nova/nova-dhcpbridge.conf	(StrOpt) location of flagfile for dhcpbridge
dmz_cidr=10.128.0.0/24	(StrOpt) dmz range that should be accepted
dns_server=[]	(MultiStrOpt) if set, uses specific dns server for dnsmasq. Can be specified multiple times.
use_network_dns_servers=False	(BoolOpt) if set, uses the dns server from the node's network settings for the servers in dnsmasq.

Configuration option=Default value	(Type) Description
dnsmasq_config_file=	(StrOpt) Override the default dnsmasq settings with this file
linuxnet_interface_driver=nova.network.linux_net.LinuxBridgeDriver	(StrOpt) Interface driver used to create ethernet devices.
linuxnet_ovs_integration_bridge=br-int	(StrOpt) Name of Open vSwitch bridge used with linuxnet
network_device_mtu=<None>	(StrOpt) MTU setting for vlan
networks_path=\$state_path/networks	(StrOpt) Location to keep network config files
public_interface=eth0	(StrOpt) Interface for public IP addresses
routing_source_ip=\$my_ip	(StrOpt) Public IP of network host
send_arp_for_ha=false	(BoolOpt) send gratuitous ARPs for HA setup
use_single_default_gateway=false	(BoolOpt) Use single default gateway. Only first nic of vm will get default gateway from dhcp server
forward_bridge_interface=all	(ListOpt) An interface that bridges can forward to. If this is set to all then all traffic will be forwarded. This is useful in vlan mode when a deployer doesn't want traffic to be routed between vlans. Note that forward_bridge_interface can be specified multiple times and should be specified once for each interface that supports floating ips.
auto_assign_floating_ip=false	(BoolOpt) Autoassigning floating IP to VM
cnt_vpnc_clients=0	(IntOpt) Number of addresses reserved for vpn clients
create_unique_mac_address_attempts=5	(IntOpt) Number of attempts to create unique mac address
default_floating_pool=nova	(StrOpt) Default pool for floating ips
dhcp_domain=nomalocal	(StrOpt) domain to use for building the hostnames
fake_call=false	(BoolOpt) If True, skip using the queue and make local calls
fixed_ip_disassociate_timeout=600	(IntOpt) Seconds after which a deallocated IP is disassociated
fixed_range=10.0.0.0/8	(StrOpt) Fixed IP address block
flat_injected=false	(BoolOpt) Whether to attempt to inject network setup into guest
flat_interface=<None>	(StrOpt) FlatDhcp will bridge into this interface if set
flat_network_bridge=<None>	(StrOpt) Bridge for simple network instances
flat_network_dns=8.8.4.4	(StrOpt) Dns for simple network
floating_range=4.4.4.0/24	(StrOpt) Floating IP address block
force_dhcp_release=false	(BoolOpt) If True, send a dhcp release on instance termination
gateway=<None>	(StrOpt) Default IPv4 gateway
l3_lib=nova.network.l3.LinuxNetL3	(StrOpt) Indicates underlying L3 management library
multi_host=false	(BoolOpt) Default value for multi_host in networks. Enable when using nova networking (not quantum) on multiple compute nodes for creation and runtime efficiency. Also provides incremental fault tolerance.
share_dhcp_address=false	(BoolOpt) If True and multi_host is also true all compute hosts will share the same dhcp address.
update_dns_entries=false	(BoolOpt) True will message all network hosts to update their DNS entries when update occurs. Useful when multi_host is True.
network_host=MGG2WEDRJM	(StrOpt) Network host to use for IP allocation in flat modes
network_size=256	(IntOpt) Number of addresses in each private subnet

Configuration option=Default value	(Type) Description
num_networks=1	(IntOpt) Number of networks to support
vlan_interface=<None>	(StrOpt) VLANs will bridge into this interface if set
vlan_start=100	(IntOpt) First VLAN for private networks
vpn_ip=\$my_ip	(StrOpt) Public IP for the cloudpipe VPN servers
vpn_start=1000	(IntOpt) First VPN port for private networks
CloudPipe specifics	
boot_script_template=\$pybasedir/nova/cloudpipe/bootscript.template	(StrOpt) Template for cloudpipe instance boot script
dmz_mask=255.255.255.0	(StrOpt) Netmask to push into openvpn config
dmz_net=10.0.0.0	(StrOpt) Network to push into openvpn config
vpn_instance_type=m1.tiny	(StrOpt) Instance type for vpn instances
Quantum specifics	
network_api_class=nova.network.api.API	(StrOpt) Defaults to nova-network. Must be modified to nova.network.quantumv2.api.API indicate that Quantum should be used rather than the traditional nova-network networking model.
quantum_url=http://127.0.0.1:9696	(IntOpt) URL for connecting to the Quantum networking service. Indicates the hostname/IP and port of the Quantum server for your deployment.
quantum_auth_strategy=keystone	(StrOpt) Should be kept as default 'keystone' for all production deployments.
quantum_admin_tenant_name=<None>	(StrOpt) Tenant name for connecting to Quantum network services in admin context through the OpenStack Identity service.
quantum_admin_username=<None>	(StrOpt) Username for connecting to Quantum network services in admin context through the OpenStack Identity service.
quantum_admin_password=<None>	(StrOpt) Password for connecting to Quantum network services in admin context through the OpenStack Identity service.
quantum_admin_auth_url=<None>	(StrOpt) Points to the OpenStack 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.
quantum_region_name=<None>	(StrOpt) Region name for connecting to quantum in admin context, through the OpenStack Identity service,
quantum_extension_sync_interval=<600>	(IntOpt) Some OpenStack Networking (quantum) extensions require Compute (nova) to pass in additional fields based on which extension the Networking (quantum) service is running. This is the number of seconds that nova will wait before requerying.

**Table 5.17. Description of nova.conf file configuration options for live migration**

Configuration option=Default value	(Type) Description
live_migration_bandwidth=0	(IntOpt) Maximum bandwidth to be used during migration transfer, in Mbps.
live_migration_flag= VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER	(StrOpt) Migration flags to be set for live migration, defines parameters for use during the migration.
live_migration_retry_count=30	(IntOpt) Number of one-second retries needed in live_migration.

Configuration option=Default value	(Type) Description
live_migration_uri=qemu+tcp:///%s/system	(StrOpt) Define Host URI used by live_migration feature. Any included "%s" is replaced with the migration target host name.

**Table 5.18. Description of nova.conf file configuration options for compute nodes**

Configuration option=Default value	(Type) Description
base_dir_name=	(StrOpt) Directory where cached images are stored under directory indicated with instances_path
compute_driver=nova.virt.connection.get_connection	(StrOpt) Driver to use for controlling virtualization
console_host=MGG2WEDRJM	(StrOpt) Console proxy host to use to connect to instances on this host.
default_notification_level=INFO	(StrOpt) Default notification level for outgoing notifications
default_publisher_id=\$host	(StrOpt) Default publisher_id for outgoing notifications
heal_instance_info_cache_interval=60	(IntOpt) Number of seconds between instance info_cache self healing updates
host_state_interval=120	(IntOpt) Interval in seconds for querying the host status
image_cache_manager_interval=40	(IntOpt) Number of periodic scheduler ticks to wait between runs of the image cache manager.
instances_path=\$state_path/instances	(StrOpt) where instances are stored on disk
reboot_timeout=0	(IntOpt) Automatically hard reboot an instance if it has been stuck in a rebooting state longer than N seconds. Set to 0 to disable.
rescue_timeout=0	(IntOpt) Automatically unrescue an instance after N seconds. Set to 0 to disable.
resize_confirm_window=0	(IntOpt) Automatically confirm resizes after N seconds. Set to 0 to disable.
running_deleted_instance_action=log	(StrOpt) Action to take if a running deleted instance is detected. Valid options are 'noop', 'log' and 'reap'. Set to 'noop' to disable.
running_deleted_instance_poll_interval=30	(IntOpt) Number of periodic scheduler ticks to wait between runs of the cleanup task.
running_deleted_instance_timeout=0	(IntOpt) Number of seconds after being deleted when a running instance should be considered eligible for cleanup.

**Table 5.19. Description of nova.conf file configuration options for bare metal deployment**

Configuration option=Default value	(Type) Description
	Options should be placed in the [baremetal] config group
db_backend=sqlalchemy	(StrOpt) The backend to use for db
deploy_kernel	(StrOpt) Glance image UUID for the special deploy kernel. Can also be set on the flavor (instance type).
deploy_ramdisk	(StrOpt) Glance image UUID for the special deploy ramdisk. Can also be set on the flavor (instance type).
driver=nova.virt.baremetal.pxe.PXE	(StrOpt) Nova class for the imaging sub-driver to use
instance_type_extra_specs	(StrOpt) Additional capabilities this baremetal compute host should advertise. Should include the "cpu_arch" of the baremetal nodes managed by this host, which must match hardware and flavor extra_specs. Example: cpu_arch:x86_64

Configuration option=Default value	(Type) Description
ipmi_power_retry=5	(IntOpt) Number of times that an IPMI command should be retried before raising an error and aborting the action. Actions are retried at half second intervals.
net_config_template=\$pybasedir/nova/virt/baremetal/net-dhcp.ubuntu.template	(StrOpt) Template file for injected network. Use net-static.ubuntu.template if you are not using Quantum DHCP.
power_manager=nova.virt.baremetal.ipmi.IPMI	(StrOpt) Nova class for the power sub-driver to use.
pxe_append_params	(StrOpt) Any additional parameters that must be passed to the baremetal nodes during the PXE boot process.
pxe_config_template=\$pybasedir/nova/virt/baremetal/pxe_config.template	(StrOpt) Template file for PXE configuration.
pxe_deploy_timeout=0	(IntOpt) Timeout in seconds to wait for PXE deployment to complete. Defaults to 0 (unlimited). This should be set to a value appropriate for each environment, but should not be more than instance_build_timeout.
sql_connection=sqlite:///\$state_path/baremetal_\$sqlite_db	(StrOpt) The SQLAlchemy connection string used to connect to the database
terminal=shellinaboxd	(StrOpt) Path to remote terminal program that provides terminal access to baremetal nodes.
terminal_cert_dir	(StrOpt) Path to directory which stores SSL/PEM certs for terminal access.
terminal_pid_dir	(StrOpt) Path to directory which stores PID files for terminal access.
tftpboot=/tftpboot	(StrOpt) Path to directory where TFTP images should be placed. Required for PXE driver.
vif_driver=nova.virt.baremetal.vif_driver.BareMetalVIFDriver	(StrOpt) Nova class for the VIF sub-driver to use.
volume_driver=nova.virt.baremetal.volume_driver.LibvirtVolumeDriver	(StrOpt) Nova class for the Volume sub-driver to use.

**Table 5.20. Description of nova.conf file configuration options for hypervisors**

Configuration option=Default value	(Type) Description
block_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PRESERVE_SOURCE,VIR_MIGRATE_NON_BLOCK,VIR_MIGRATE_BLOCK,VIR_MIGRATE_PRESERVE_BLOCK,VIR_MIGRATE_NON_SHARED_INC	(StrOpt) Define migration flags for block migration
checksum_base_images=false	(BoolOpt) Used as an additional check to detect if cached images have become corrupted. If true, the compute service will write checksums for image files in the /var/lib/nova/instances/_base directory to disk, and do periodic checks to verify that this checksum is valid. If the checksum fails to validate, the failure is recorded to the log as an error, but no other action is taken: it is assumed that an operator will monitor the logs and take appropriate action. If false, image hashes are not verified.
hyperv_attaching_volume_retry_count=10	(IntOpt) Number of times to retry attaching to a volume when using the Hyper-V hypervisor
hyperv_wait_between_attach_retry=5	(IntOpt) To be written: found in /nova/virt/hyperv/volumeops.py
libvirt_cpu_mode=<None>	(StrOpt) Configures the guest CPU model exposed to the hypervisor. Valid options are: custom, host-model, host-passthrough, none. If the hypervisor is KVM or QEMU, the default value is host-model, otherwise the default value is none.
libvirt_cpu_model=<None>	(StrOpt) Specify the guest CPU model exposed to the hypervisor. This configuration option is only applicable if libvirt_cpu_mode is set to custom. Valid options: one of the named models specified in /usr/share/libvirt/cpu_map.xml, e.g.: Westmere, Nehalem, Opteron_G3.

Configuration option=Default value	(Type) Description
libvirt_disk_prefix=<None>	(StrOpt) Override the default disk prefix for the devices attached to a server, which is dependent on libvirt_type. (valid options are: sd, xvd, uvd, vd)
libvirt_inject_key=true	(BoolOpt) Inject the ssh public key at boot time
libvirt_inject_partition=1	(IntOpt) The partition to inject to : -2 => disable, -1 => inspect (libguestfs only), 0 => not partitioned, >0 => partition number'
libvirt_images_type=default	(StrOpt) Instance ephemeral storage backend format. Acceptable values are: raw, qcow2, lvm, default. If default is specified, then use_cow_images flag is used instead of this one. Please note, that current snapshot mechanism in OpenStack Compute works only with instances backed with Qcow2 images.
libvirt_images_volume_group=None	(StrOpt) LVM Volume Group that is used for instance ephemerals, when you specify libvirt_images_type=lvm.
libvirt_inject_password=false	(BoolOpt) Inject the admin password at boot time, without an agent.
libvirt_lvm_snapshot_size=1000	(IntOpt) The amount of storage (in megabytes) to allocate for LVM snapshot copy-on-write blocks.
libvirt_nonblocking=true	(BoolOpt) Use a separated OS thread pool to realize non-blocking libvirt calls
libvirt_snapshots_directory=\$instances_path/snapshots	(StrOpt) Location where libvirt driver will store snapshots before uploading them to image service
libvirt_snapshot_compression=False	(BoolOpt) Compresses snapshot images when possible. This currently applies exclusively to qcow2 images.
libvirt_sparse_logical_volumes=false	(BoolOpt) Create sparse (not fully allocated) LVM volumes for instance ephemerals if you use LVM backend for them.
libvirt_type=kvm	(StrOpt) Libvirt domain type (valid options are: kvm, lxc, qemu, uml, xen)
libvirt_uri=	(StrOpt) Override the default libvirt URI (which is dependent on libvirt_type)
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtGenericVifDriver	(StrOpt) The libvirt VIF driver to configure the VIFs.
libvirt_volume_drivers="iscsi=nova.virt.libvirt.volume.LibvirtISCSIDriver, local=nova.virt.libvirt.volume.LibvirtVolumeDriver, fake=nova.virt.libvirt.volume.LibvirtFakeVolumeDriver, rbd=nova.virt.libvirt.volume.LibvirtNetVolumeDriver, sheepdog=nova.virt.libvirt.volume.LibvirtNetVolumeDriver, glusterfs=nova.virt.libvirt.volume.LibvirtGlusterfsVolumeDriver"	(StrOpt) Libvirt volume drivers for remote volumes.
libvirt_wait_soft_reboot_seconds=120	(IntOpt) Number of seconds to wait for instance to shutdown after soft reboot request is made. We fall back to hard reboot if instance does not shutdown within this window.
limit_cpu_features=false	(BoolOpt) Used by Hyper-V
remove_unused_base_images=true	(BoolOpt) Indicates whether unused base images should be removed
remove_unused_kernels=false	(BoolOpt) Should unused kernel images be removed? If unused images should be removed set to true, if not, set to false. This option is only safe to set to true if all compute nodes have been updated to support this option so that older image cache managers on remote compute nodes are prevented from cleaning up kernels because

Configuration option=Default value	(Type) Description
	they appear unused. This will be enabled by default in a future release.
remove_unused_original_minimum_age_seconds=86400	(IntOpt) Unused unresized base images younger than this will not be removed
remove_unused_resized_minimum_age_seconds=3600	(IntOpt) Unused resized base images younger than this will not be removed
rescue_image_id=<None>	(StrOpt) Rescue ami image
rescue_kernel_id=<None>	(StrOpt) Rescue aki image
rescue_ramdisk_id=<None>	(StrOpt) Rescue ari image
snapshot_image_format=<None>	(StrOpt) Snapshot image format (valid options are : raw, qcow2, vmdk, vdi). Defaults to same as source image
use_usb_tablet=true	(BoolOpt) Sync virtual and real mouse cursors in Windows VMs
libvirt integration	
libvirt_ovs_bridge=br-int	(StrOpt) Name of Integration Bridge used by Open vSwitch
libvirt_use_virtio_for_bridges=false	(BoolOpt) Use virtio for bridge interfaces
VMWare integration	
vmwareapi_wsdl_loc=<None>	(StrOpt) VIM Service WSDL Location e.g http://<server>/vimService.wsdl, due to a bug in vSphere ESX 4.1 default wsdl.
vmware_vif_driver=nova.virt.vmwareapi.vif.VMWareVlanBridgeDriver	(Type Opt) The VMWare VIF driver to configure the VIFs.
vmwareapi_api_retry_count=10	(FloatOpt) The number of times we retry on failures, e.g., socket error, etc. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_ip=<None>	(StrOpt) URL for connection to VMWare ESX host. Required if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_password=<None>	(StrOpt) Password for connection to VMWare ESX host. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_username=<None>	(StrOpt) Username for connection to VMWare ESX host. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_task_poll_interval=5.0	(FloatOpt) The interval used for polling of remote tasks. Used only if compute_driver is vmwareapi.VMwareESXDriver,
vmwareapi_vlan_interface=vmnic0	(StrOpt) Physical ethernet adapter name for vlan networking
powervm_mgr_type=ivm	(StrOpt) PowerVM system manager type (ivm, hmc)
powervm_mgr=<None>	(StrOpt) PowerVM manager host or ip
powervm_vios=powervm_mgr	(StrOpt) PowerVM VIOS host or ip if different from manager
powervm_mgr_user=<None>	(StrOpt) PowerVM manager user name
powervm_mgr_passwd=<None>	(StrOpt) PowerVM manager user password
powervm_img_remote_path=<None>	(StrOpt) PowerVM image remote path. Used to copy and store images from Glance on the PowerVM VIOS LPAR.
powervm_img_local_path=<None>	(StrOpt) Local directory on the compute host to download glance images to.

**Table 5.21. Description of nova.conf file configuration options for console access to VMs on VMWare VMRC or XenAPI**

<b>Configuration option=Default value</b>	<b>(Type) Description</b>
console_driver=nova.console.xvp.XVPCConsoleProxy	(StrOpt) Driver to use for the console proxy
console_public_hostname=MGG2WEDRJM	(StrOpt) Publicly visible name for this console host
stub_compute=false	(BoolOpt) Stub calls to compute worker for tests
console_vmrc_error_retries=10	(IntOpt) number of retries for retrieving VMRC information
console_vmrc_port=443	(IntOpt) port for VMware VMRC connections
console_xvp_conf=/etc/xvp.conf	(StrOpt) generated XVP conf file
console_xvp_conf_template=\$pybasedir/nova/console/xvp.conf.template	(StrOpt) XVP conf template
console_xvp_log=/var/log/xvp.log	(StrOpt) XVP log file
console_xvp_multiplex_port=5900	(IntOpt) port for XVP to multiplex VNC connections on
console_xvp_pid=/var/run/xvp.pid	(StrOpt) XVP master process pid file
xenapi_agent_path=usr/sbin/xe-update-networking	(StrOpt) Specifies the path in which the xenapi guest agent should be located. If the agent is present, network configuration is not injected into the image. Used if compute_driver=xenapi.XenAPIDriver and flat_injected=True.
xenapi_connection_concurrent=5	(IntOpt) Maximum number of concurrent XenAPI connections. Used only if compute_driver=xenapi.XenAPIDriver.
xenapi_connection_url=<None>	(StrOpt) URL for connection to XenServer/Xen Cloud Platform. Required if compute_driver=xenapi.XenAPIDriver.
xenapi_connection_username=root	(StrOpt) Password for connection to XenServer/Xen Cloud Platform. Used only if compute_driver=xenapi.XenAPIDriver.
xenapi_connection_password=<None>	(StrOpt) Username for connection to XenServer/Xen Cloud Platform. Used only if compute_driver=xenapi.XenAPIDriver.
xenapi_check_host=true	(BoolOpt) Ensure compute service is running on host XenAPI connects to.
xenapi_login_timeout=10	(BoolOpt) Timeout in seconds for XenAPI login.
xenapi_remap_vbd_dev=false	(BoolOpt) Used to enable the remapping of VBD dev. (Works around an issue in Ubuntu Maverick).
xenapi_remap_vbd_dev_prefix=sd	(StrOpt) Specify prefix to remap VBD dev to (ex. /dev/xvdb -> /dev/sdb). Used when xenapi_remap_vbd_dev=true.
xenapi_sr_base_path=/var/run/sr-mount	(StrOpt) Base path to the storage repository.
xenapi_vhd_coalesce_poll_interval=5.0	(FloatOpt) The interval used for polling of coalescing vhds. Used only if compute_driver=xenapi.XenAPIDriver.
xenapi_vhd_coalesce_max_attempts=5	(IntOpt) Max number of times to poll for VHD to coalesce. Used only if compute_driver=xenapi.XenAPIDriver.

**Table 5.22. Description of nova.conf file configuration options for S3 access to image storage**

<b>Configuration option=Default value</b>	<b>(Type) Description</b>
image_decryption_dir=/tmp	(StrOpt) parent dir for tempdir used for image decryption
s3_access_key=notchecked	(StrOpt) access key to use for s3 server for images

Configuration option=Default value	(Type) Description
s3_affix_tenant=false	(BoolOpt) whether to affix the tenant id to the access key when downloading from s3
s3_secret_key=notchecked	(StrOpt) secret key to use for s3 server for images
s3_use_ssl=false	(BoolOpt) whether to use ssl when talking to s3

**Table 5.23. Description of nova.conf file configuration options for schedulers that use algorithms to assign VM launch on particular compute hosts**

Configuration option=Default value	(Type) Description
scheduler_host_manager=nova.scheduler.host_manager.HostManager	(StrOpt) The scheduler host manager class to use.
scheduler_max_attempts=3	(IntOpt) Maximum number of attempts to schedule an instance before giving up and setting the instance to error.
cpu_allocation_ratio=16.0	(FloatOpt) Virtual CPU to Physical CPU allocation ratio.
ram_allocation_ratio=1.5	(FloatOpt) virtual ram to physical ram allocation ratio.
reserved_host_disk_mb=0	(IntOpt) Amount of disk in MB to reserve for host/dom0.
reserved_host_memory_mb=512	(IntOpt) Amount of memory in MB to reserve for host/dom0.
scheduler_available_filters=nova.scheduler.filters.all_filters	(MultiStrOpt) Filter classes available to the scheduler which may be specified more than once. An entry of "nova.scheduler.filters.all_filters" maps to all filters included with nova.
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,CoreFilter	(ListOpt) Which filter class names to use for filtering hosts when not specified in the request.
compute_fill_first_cost_fn_weight=-1.0	(FloatOpt) How much weight to give the fill-first cost function. A negative value will reverse behavior: e.g. spread-first.
retry_host_cost_fn_weight=1.0	(FloatOpt) How much weight to give the retry host cost function. A negative value will reverse behavior: e.g. use multiple-times-retried hosts first.
least_cost_functions=nova.scheduler.least_cost.compute_fill	(ListOpt) Which cost functions the LeastCostScheduler should use.
noop_cost_fn_weight=1.0	(FloatOpt) How much weight to give the noop cost function.
scheduler_driver=nova.scheduler.multi.MultiScheduler	(StrOpt) Default driver to use for the scheduler.
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler	(FilterScheduler) Driver to use for scheduling Compute calls.
volume_scheduler_driver=nova.scheduler.chance.ChanceScheduler	(ChanceScheduler) Driver to use for scheduling volume calls.
scheduler_json_config_location=	(StrOpt) Absolute path to scheduler configuration JSON file.
max_cores=16	(IntOpt) maximum number of instance cores to allow per host.
max_gigabytes=10000	(IntOpt) maximum number of volume gigabytes to allow per host.
max_networks=1000	(IntOpt) maximum number of networks to allow per host.
skip_isolated_core_check=true	(BoolOpt) Allow overcommitting vcpus on isolated hosts.
scheduler_host_subset_size=1	(IntOpt) New instances will be scheduled on a host chosen randomly from a subset of the N best hosts, rather than just the host with the highest weight. This property defines the subset size that a host is chosen from. A value of 1 chooses the first host returned by the weighing functions. This value must be at least 1. Any value less than 1 will be ignored, and 1 will be used instead.

**Table 5.24. Description of nova.conf file configuration options for config drive features**

Configuration option=Default value	(Type) Description
config_drive_format=	(StrOpt) File format for the config drive, options: iso9660, vfat
config_drive_skip_versions=1.0 2007-01-19 2007-03-01 2007-08-29 2007-10-10 2007-12-15 2008-02-01 2008-09-01	(StrOpt) Version numbers of the config drive releases to skip
config_drive_tempdir=<None>	(StrOpt)
force_config_drive=	(StrOpt) Forces use of config drive, such as using config drive based on image properties; reserved for future use
using_config_drive=	(StrOpt) Enables use of config drive for images launched

**Table 5.25. Description of nova.conf file configuration options for volumes attached to VMs**

Configuration option=Default value	(Type) Description
cinder_cross_az_attach=True	(BoolOpt) Allow attach between instance and volume in different availability zones.
iscsi_helper=ietadm	(StrOpt) iscsi target user-land tool to use
iscsi_ip_address=\$my_ip	(StrOpt) use this ip for iscsi
iscsi_num_targets=100	(IntOpt) Number of iscsi target ids per host
iscsi_port=3260	(IntOpt) The port that the iSCSI daemon is listening on
iscsi_target_prefix=iqn.2010-10.org.openstack:	(StrOpt) prefix for iscsi volumes
num_iscsi_scan_tries=3	(StrOpt) number of times to rescan iSCSI target to find volume
num_shell_tries=3	(StrOpt) number of times to attempt to run flakey shell commands
rbd_pool=rbd	(StrOpt) the RADOS pool in which rbd volumes are stored
rbd_secret_uuid=<None>	(StrOpt) the libvirt uuid of the secret for the rbd_uservolumes
rbd_user=<None>	(StrOpt) the RADOS client name for accessing rbd volumes
volume_group=nova-volumes	(StrOpt) Name for the VG that will contain exported volumes
netapp_login=<None>	(StrOpt) User name for the DFM server
netapp_password=<None>	(StrOpt) Password for the DFM server
netapp_server_hostname=<None>	(StrOpt) Hostname for the DFM server
netapp_server_port=8088	(IntOpt) Port number for the DFM server
netapp_storage_service=<None>	(StrOpt) Storage service to use for provisioning
netapp_vfiler=<None>	(StrOpt) Vfiler to use for provisioning
netapp_wsdl_url=<None>	(StrOpt) URL of the WSDL file for the DFM server
nexenta_blocksize=	(StrOpt) block size for volumes (blank=default,8KB)
nexenta_host=	(StrOpt) IP address of Nexenta SA
nexenta_iscsi_target_portal_port=3260	(IntOpt) Nexenta target portal port
nexenta_password=nexenta	(StrOpt) Password to connect to Nexenta SA
nexenta_rest_port=2000	(IntOpt) HTTP port to connect to Nexenta REST API server
nexenta_rest_protocol=auto	(StrOpt) Use http or https for REST connection (default auto)
nexenta_sparse=false	(BoolOpt) flag to create sparse volumes
nexenta_target_group_prefix=nova/	(StrOpt) prefix for iSCSI target groups on SA

Configuration option=Default value	(Type) Description
nexenta_target_prefix=iqn.1986-03.com.sun:02:nova-	(StrOpt) IQN prefix for iSCSI targets
nexenta_user=admin	(StrOpt) User name to connect to Nexenta SA
nexenta_volume=nova	(StrOpt) pool on SA that will hold all volumes
san_clustername=	(StrOpt) Cluster name to use for creating volumes
san_ip=	(StrOpt) IP address of SAN controller
san_is_local=false	(BoolOpt) Execute commands locally instead of over SSH; use if the volume service is running on the SAN device
san_login=admin	(StrOpt) Username for SAN controller
san_password=	(StrOpt) Password for SAN controller
san_private_key=	(StrOpt) Filename of private key to use for SSH authentication
san_ssh_port=22	(IntOpt) SSH port to use with SAN
san_thin_provision=true	(BoolOpt) Use thin provisioning for SAN volumes?
san_zfs_volume_base=rpool/	(StrOpt) The ZFS path under which to create zvols for volumes.

# 6. Identity Management

## Table of Contents

Basic Concepts .....	84
Memcached and System Time .....	90
SSL and Keystone Configuration .....	90
User CRUD .....	91
Configuration Files .....	92
Logging .....	93
Monitoring .....	93
Certificates for PKI .....	94
Sample Configuration Files .....	96
Running .....	97
Initializing Keystone .....	97
Adding Users, Tenants, and Roles with python-keystoneclient .....	97
Configuring Services to work with Keystone .....	103
Configuring Keystone SSL support .....	112
Using External Authentication with OpenStack Identity .....	113
Troubleshooting Identity (Keystone) .....	114

The default identity management system for OpenStack is the OpenStack Identity Service, code-named Keystone. Once Identity is installed, it is configured via a primary configuration file (`etc/keystone.conf`), possibly a separate logging configuration file, and initializing data into keystone using the command line client.

## Basic Concepts

The Identity service has two primary functions:

1. User management: keep track of users and what they are permitted to do
2. Service catalog: Provide a catalog of what services are available and where their API endpoints are located

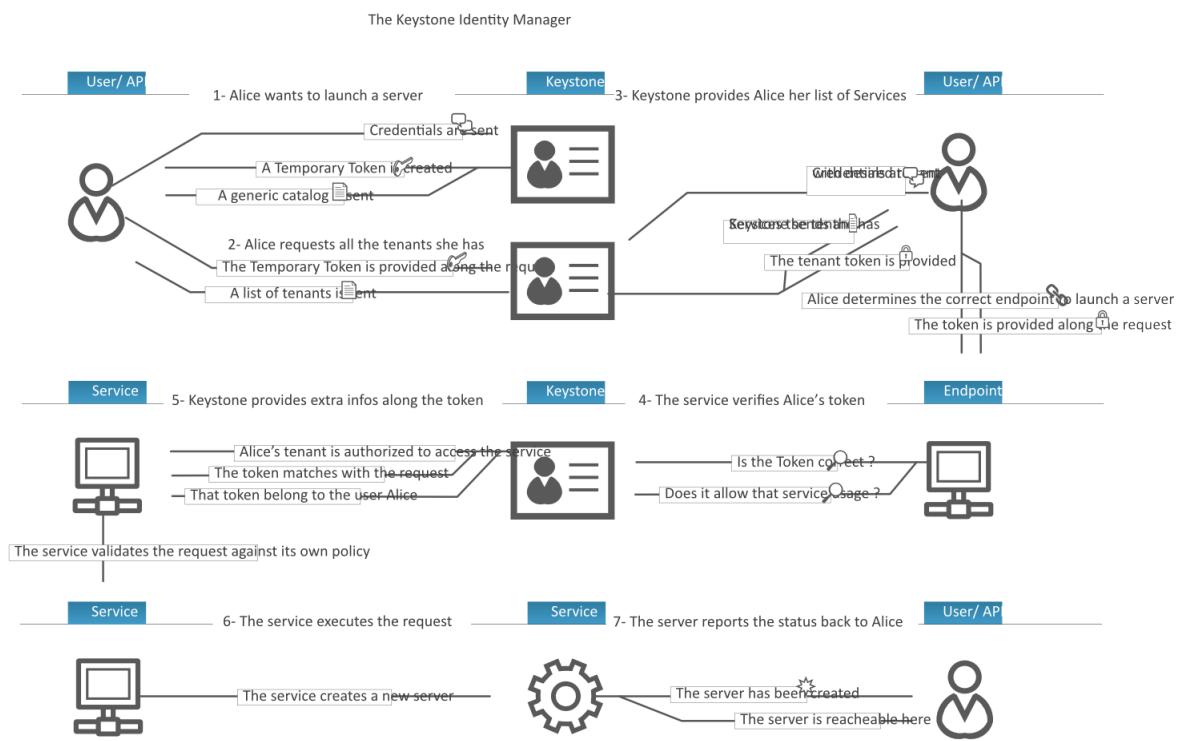
The Identity Service has several definitions which are important to understand.

User	A digital representation of a person, system, or service who uses OpenStack cloud services. Identity authentication services will validate that incoming requests are being made by the user who claims to be making the call. Users have a login and may be assigned tokens to access resources. Users may be directly assigned to a particular tenant and behave as if they are contained in that tenant.
Credentials	Data that belongs to, is owned by, and generally only known by a user that the user can present to prove they are who they are (since nobody else should know that data).

Examples are:

- a matching username and password
- a matching username and API key
- yourself and a driver's license with a picture of you
- a token that was issued to you that nobody else knows of

Authentication	In the context of the identity service, authentication is the act of confirming the identity of a user or the truth of a claim. The identity service will confirm that incoming requests are being made by the user who claims to be making the call by validating a set of claims that the user is making. These claims are initially in the form of a set of credentials (username & password, or username and API key). After initial confirmation, the identity service will issue the user a token which the user can then provide to demonstrate that their identity has been authenticated when making subsequent requests.
Token	A token is an arbitrary bit of text that is used to access resources. Each token has a scope which describes which resources are accessible with it. A token may be revoked at anytime and is valid for a finite duration.  While the identity service supports token-based authentication in this release, the intention is for it to support additional protocols in the future. The intent is for it to be an integration service foremost, and not aspire to be a full-fledged identity store and management solution.
Tenant	A container used to group or isolate resources and/or identity objects. Depending on the service operator, a tenant may map to a customer, account, organization, or project.
Service	An OpenStack service, such as Compute (Nova), Object Storage (Swift), or Image Service (Glance). A service provides one or more endpoints through which users can access resources and perform (presumably useful) operations.
Endpoint	An network-accessible address, usually described by URL, where a service may be accessed. If using an extension for templates, you can create an endpoint template, which represents the templates of all the consumable services that are available across the regions.
Role	A personality that a user assumes when performing a specific set of operations. A role includes a set of rights and privileges. A user assuming that role inherits those rights and privileges.  In the identity service, a token that is issued to a user includes the list of roles that user can assume. Services that are being called by that user determine how they interpret the set of roles a user has and which operations or resources each roles grants access to.



## User management

The three main concepts of Identity user management are:

- Users
- Tenants
- Roles

A *user* represents a human user, and has associated information such as username, password and email. This example creates a user named "alice":

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

A *tenant* can be thought of as a project, group, or organization. Whenever you make requests to OpenStack services, you must specify a tenant. For example, if you query the Compute service for a list of running instances, you will receive a list of all of the running instances in the tenant you specified in your query. This example creates a tenant named "acme":

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



### Note

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

A *role* captures what operations a user is permitted to perform in a given tenant. This example creates a role named "compute-user":

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



### Note

It is up to individual services such as the Compute service and Image service to assign meaning to these roles. As far as the Identity service is concerned, a role is simply a name.

The Identity service associates a user with a tenant and a role. To continue with our previous examples, we may wish to assign the "alice" user the "compute-user" role in the "acme" tenant:

```
$ keystone user-list

+-----+-----+-----+-----+
| id | enabled | email | name |
+-----+-----+-----+-----+
| 892585 | True | alice@example.com | alice |
+-----+-----+-----+-----+

$ keystone role-list

+-----+-----+
| id | name |
+-----+-----+
| 9a764e | compute-user |
+-----+-----+

$ keystone tenant-list

+-----+-----+-----+
| id | name | enabled |
+-----+-----+-----+
| 6b8fd2 | acme | True |
+-----+-----+-----+

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

A user can be assigned different roles in different tenants: for example, Alice may also have the "admin" role in the "Cyberdyne" tenant. A user can also be assigned multiple roles in the same tenant.

The `/etc/[SERVICE_CODENAME]/policy.json` controls what users are allowed to do 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 will be 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 will be able to create volumes in that tenant.

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

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

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

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

```
{
    "admin_or_owner": [[{"role:admin"}, {"project_id: %(project_id)s"}]],
    "default": [[{"rule:admin_or_owner"}]],

    "compute:create": [{"role": "compute-user"}],
    "compute:create:attach_network": [{"role": "compute-user"}],
    "compute:create:attach_volume": [{"role": "compute-user"}],
    "compute:get_all": [{"role": "compute-user"}],

    "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_api"}]],
    "compute_extension:admin_actions:unlock": [[{"rule:admin_api"}]],
    "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: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 two main concepts of Identity service management are:

- Services
- Endpoints

The Identity service also maintains a user that corresponds to each service (e.g., a user named *nova*, for the Compute service) and a special service tenant, which is called *service*.

The commands for creating services and endpoints are described in a later section.

## Memcached and System Time

If using [memcached](#) with Keystone - e.g. using the memcache token driver or the `auth_token` middleware - ensure that the system time of memcached hosts is set to UTC. Memcached uses the host's system time in determining whether a key has expired, whereas Keystone sets key expiry in UTC. The timezone used by Keystone and memcached must match if key expiry is to behave as expected.

## SSL and Keystone Configuration

Keystone may be configured to support 2-way SSL out-of-the-box. The x509 certificates used by Keystone must be obtained externally and configured for use with Keystone as described in this section. However, a set of sample certificates is provided in the `examples/pki/certs` and `examples/pki/private` directories with the Keystone distribution for testing. Here is the description of each of them and their purpose:

## Types of certificates

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

Note that you may choose whatever names you want for these certificates, or combine the public/private keys in the same file if you wish. These certificates are just provided as an example.

## SSL Configuration

To enable SSL with client authentication, modify the etc/keystone.conf file accordingly under the [ssl] section. SSL configuration example using the included sample certificates:

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

- **enable:** True enables SSL. Defaults to False.
- **certfile:** Path to Keystone public certificate file.
- **keyfile:** Path to Keystone private certificate file. If the private key is included in the certfile, the keyfile maybe omitted.
- **ca\_certs:** Path to CA trust chain.
- **cert\_required:** Requires client certificate. Defaults to False.

## User CRUD

Keystone provides a user CRUD filter that can be added to the public\_api pipeline. This user crud filter allows 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 app in the public\_api WSGI pipeline in keystone.conf e.g.:

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

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

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

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

In addition to changing their password all of the users current tokens will be deleted (if the backend used is kvs or sql)

## Configuration Files

The Identity configuration file is an 'ini' file format with sections, extended from [Paste](#), a common system used to configure python WSGI based applications. In addition to the paste config entries, general configuration values are stored under [DEFAULT], [sql], [ec2] and then drivers for the various services are included under their individual sections.

The services include:

- [DEFAULT] - general configuration
- [sql] - optional storage backend configuration
- [ec2] - Amazon EC2 authentication driver configuration
- [s3] - Amazon S3 authentication driver configuration.
- [identity] - identity system driver configuration
- [catalog] - service catalog driver configuration
- [token] - token driver configuration
- [policy] - policy system driver configuration for RBAC
- [signing] - cryptographic signatures for PKI based tokens
- [ssl] - SSL configuration

The configuration file is expected to be named `keystone.conf`. When starting Identity, you can specify a different configuration file to use with `--config-file`. If you do **not** specify a configuration file, keystone will look in the following directories for a configuration file, in order:

- `~/.keystone`
- `~/`
- `/etc/keystone`

- /etc

## Logging

Logging is configured externally to the rest of Identity, the file specifying the logging configuration is in the [DEFAULT] section of the `keystone.conf` file under `log_config`. If you wish to route all your logging through syslog, set `use_syslog=true` option in the [DEFAULT] section.

A sample logging file is available with the project in the directory `etc/logging.conf.sample`. Like other OpenStack projects, Identity uses the `python logging module`, which includes extensive configuration options for choosing the output levels and formats.

In addition to this documentation page, you can check the `etc/keystone.conf.sample` configuration files distributed with keystone for example configuration files for each server application.

For services which have separate paste-deploy ini file, auth\_token middleware can be alternatively configured in [keystone\_auth\_token] section in the main config file, such as `nova.conf`. For example in Nova, all middleware parameters can be removed from `api-paste.ini` like these:

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

and set in `nova.conf` like these:

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

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

Note that middleware parameters in paste config take priority, they must be removed to use values in [keystone\_auth\_token] section.

## Monitoring

Keystone provides some basic request/response monitoring statistics out of the box.

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

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

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

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

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

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

Query the admin API for statistics using:

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

Reset collected data using:

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

## Certificates for PKI

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



### 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-mange pki_setup` command, your best option is to run as the `pki` user. If you run `nova-manage` as root, you can append `--keystone-user` and `--keystone-group` parameters to set the username and group keystone is going to run under.

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

- `token_format` - Determines the algorithm used to generate tokens. Can be either `UUID` or `PKI`. Defaults to `PKI`.

- `certfile` - Location of certificate used to verify tokens. Default is `/etc/keystone/ssl/certs/signing_cert.pem`.
- `keyfile` - Location of private key used to sign tokens. Default is `/etc/keystone/ssl/private/signing_key.pem`.
- `ca_certs` - Location of certificate for the authority that issued the above certificate. Default is `/etc/keystone/ssl/certs/ca.pem`.
- `key_size` - Default is 1024.
- `valid_days` - Default is 3650.
- `ca_password` - Password required to read the `ca_file`. Default is `None`.

## Signing Certificate Issued by External CA

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

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

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

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

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

## Request Signing Certificate from 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.

First create a certificate request configuration file (e.g. `cert_req.conf`):

```
[ req ]  
default_bits      = 1024  
default_keyfile   = keystonekey.pem  
default_md        = sha1  
  
prompt            = no  
distinguished_name = distinguished_name  
  
[ distinguished_name ]
```

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

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

For example:

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

If everything is successfully, 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 External Signing Certificate

Assuming you have the following already:

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

Copy the above to your certificate directory. For example:

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



### Note

Make sure the certificate directory is only accessible by root.

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

## Sample Configuration Files

- `etc/keystone.conf`
- `etc/logging.conf.sample`

## Running

Running Identity is simply starting the services by using the command:

```
keystone-all
```

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

## Initializing Keystone

`keystone-manage` is designed to execute commands that cannot be administered through the normal REST api. At the moment, the following calls are supported:

- `db_sync`: Sync the database.
- `import_nova_auth`: Load auth data from a dump created with `keystone-manage`.

Generally, the following is the first step after a source installation:

```
keystone-manage db_sync
```

Invoking `keystone-manage` by itself will give you additional usage information.

## Adding Users, Tenants, and Roles with python-keystoneclient

User, tenants, and roles must be administered using admin credentials. There are two ways to configure `python-keystoneclient` to use admin credentials, using the token auth method, or password auth method.

### Token Auth Method

To use keystone client using token auth, set the following flags

- `--endpoint SERVICE_ENDPOINT` : allows you to specify the keystone endpoint to communicate with. The default endpoint is `http://localhost:35357/v2.0`'
- `--token SERVICE_TOKEN` : your administrator service token.

### Password Auth Method

- `--username OS_USERNAME` : allows you to specify the administrator username
- `--password OS_PASSWORD` : Your administrator password

- --tenant\_name OS\_TENANT\_NAME : Name of your tenant
- --auth\_url OS\_AUTH\_URL : url of your keystone auth server, for example <http://localhost:5000/v2.0>'

## 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 SERVICE_ENDPOINT=http://127.0.0.1:5000/v2.0/
export SERVICE_TOKEN=secrete_token
keystone user-list
keystone tenant-create --name=demo

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

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

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

## Tenants

Tenants are the high level grouping within Keystone that represent groups of users. A tenant is the grouping that owns virtual machines within Nova, or containers within Swift. A tenant can have zero or more users, Users can be associated with more than one tenant, and each tenant - user pairing can have a role associated with it.

### tenant-create

keyword arguments

- name
- description (optional, defaults to None)
- enabled (optional, defaults to True)

example:

```
keystone tenant-create --name=demo
```

creates a tenant named "demo".

## tenant-delete

arguments

- tenant\_id

example:

```
keystone tenant-delete f2b7b39c860840dfa47d9ee4adffa0b3
```

## tenant-enable

arguments

- tenant\_id

example:

```
keystone tenant-enable f2b7b39c860840dfa47d9ee4adffa0b3
```

## tenant-disable

arguments

- tenant\_id

example:

```
keystone tenant-disable f2b7b39c860840dfa47d9ee4adffa0b3
```

# Users

## user-create

keyword arguments:

- name
- pass
- email
- default\_tenant (optional, defaults to None)
- enabled (optional, defaults to True)

example:

```
keystone user-create  
--name=admin \  
--pass=secrete \  
--email=admin@example.com
```

## user-delete

keyword arguments:

- user

example:

```
keystone user-delete f2b7b39c860840dfa47d9ee4adffa0b3
```

## user-list

list users in the system, optionally by a specific tenant (identified by tenant\_id)

arguments

- tenant\_id (optional, defaults to None)

example:

```
keystone user-list
```

## user-update --email

arguments

- user\_id
- email

example:

```
keystone user-update --email 03c84b51574841ba9a0d8db7882ac645  
"someone@somewhere.com"
```

## user-enable

arguments

- user\_id

example:

```
keystone user-enable 03c84b51574841ba9a0d8db7882ac645
```

## **user-disable**

arguments

- user\_id

example:

```
keystone user-disable 03c84b51574841ba9a0d8db7882ac645
```

## **user-update --password**

arguments

- user\_id
- password

example:

```
keystone user-update --password 03c84b51574841ba9a0d8db7882ac645 foo
```

# **Roles**

## **role-create**

arguments

- name

example:

```
keystone role-create --name=demo
```

## **role-delete**

arguments

- role\_id

example:

```
keystone role-delete 19d1d3344873464d819c45f521ff9890
```

## **role-list**

example:

```
keystone role-list
```

## role-get

arguments

- role\_id

example:

```
keystone role-get role=19d1d3344873464d819c45f521ff9890
```

## add-user-role

arguments

- role\_id
- user\_id
- tenant\_id

example:

```
keystone add-user-role \  
3a751f78ef4c412b827540b829e2d7dd \  
03c84b51574841ba9a0d8db7882ac645 \  
20601a7f1d94447daaa4dff438cb1c209
```

## remove-user-role

arguments

- role\_id
- user\_id
- tenant\_id

example:

```
keystone remove-user-role \  
19d1d3344873464d819c45f521ff9890 \  
08741d8ed88242ca88d1f61484a0fe3b \  
20601a7f1d94447daaa4dff438cb1c209
```

# Services

## service-create

keyword arguments

- name
- type
- description

example:

```
keystone service create \
--name=nova \
--type=compute \
--description="Nova Compute Service"
```

## service-list

arguments

- service\_id

example:

```
keystone service-list
```

## service-get

arguments

- service\_id

example:

```
keystone service-get 08741d8ed88242ca88d1f61484a0fe3b
```

## service-delete

arguments

- service\_id

example:

```
keystone service-delete 08741d8ed88242ca88d1f61484a0fe3b
```

# Configuring Services to work with Keystone

Once Keystone is installed and running, services need to be configured to work with it. To do this, we primarily install and configure middleware for the OpenStack service to handle authentication tasks or otherwise interact with Keystone.

In general:

- Clients making calls to the service will pass in an authentication token.
- The Keystone middleware will look for and validate that token, taking the appropriate action.
- It will also retrieve additional information from the token such as user name, id, tenant name, id, roles, etc...

The middleware will pass those data down to the service as headers.

## Setting up credentials

To ensure services that you add to the catalog know about the users, tenants, and roles, you must create an admin token and create service users. These sections walk through those requirements.

### Admin Token

For a default installation of Keystone, before you can use the REST API, you need to define an authorization token. This is configured in `keystone.conf` file under the section `[DEFAULT]`. In the sample file provided with the keystone project, the line defining this token is

```
[DEFAULT] admin_token = ADMIN
```

This configured token is a "shared secret" between keystone and other OpenStack services, and is used by the client to communicate with the API to create tenants, users, roles, etc.

### Setting up tenants, users, and roles

You need to minimally define a tenant, user, and role to link the tenant and user as the most basic set of details to get other services authenticating and authorizing with keystone.

You will also want to create service users for Compute (nova), Image (glance), Object Storage (swift), etc. to be able to use to authenticate users against the Identity service (keystone). The `auth_token` middleware supports using either the shared secret described above as ``admin_token`` or users for each service.

See the [configuration section](#) for a walk through on how to create tenants, users, and roles.

## Setting up services

### Creating Service Users

To configure the OpenStack services with service users, we need to create a tenant for all the services, and then users for each of the services. We then assign those service users an Admin role on the service tenant. This allows them to validate tokens - and authenticate and authorize other user requests.

Create a tenant for the services, typically named 'service' (however, the name can be whatever you choose):

```
keystone tenant-create --name=service
```

This returns a UUID of the tenant - keep that, you'll need it when creating the users and specifying the roles.

Create service users for nova, glance, swift, and quantum (or whatever subset is relevant to your deployment):

```
keystone user-create --name=nova \
    --pass=Sekr3tPass \
    --tenant_id=[the uuid of the tenant] \
    --email=nova@nothing.com
```

Repeat this for each service you want to enable. Email is a required field in keystone right now, but not used in relation to the service accounts. Each of these commands will also return a UUID of the user. Keep those to assign the Admin role.

For adding the Admin role to the service accounts, you'll need to know the UUID of the role you want to add. If you don't have them handy, you can look it up quickly with:

```
keystone role-list
```

Once you have it, assign the service users to the Admin role. This is all assuming that you've already created the basic roles and settings as described in the configuration section:

```
keystone user-role-add --tenant_id=[uuid of the service tenant] \
    --user=[uuid of the service account] \
    --role=[uuid of the Admin role]
```

## Defining Services

Keystone also acts as a service catalog to let other OpenStack systems know where relevant API endpoints exist for OpenStack Services. The OpenStack Dashboard, in particular, uses this heavily - and this **must** be configured for the OpenStack Dashboard to properly function.

The endpoints for these services are defined in a template, an example of which is in the project as the file `etc/default_catalog.templates`. When keystone uses a template file backend, then changes made to the endpoints are kept in memory and don't persist if you restart the service or reboot the machine. Use the SQL backend when deploying a system for production.

Keystone supports two means of defining the services, one is the catalog template, as described above - in which case everything is detailed in that template.

The other is a SQL backend for the catalog service, in which case after keystone is online, you need to add the services to the catalog:

```
keystone service-create --name=nova \
    --type=compute \
```

```
--description="Nova Compute Service"
keystone service-create --name=ec2 \
                      --type=ec2 \
                      --description="EC2 Compatibility Layer"
keystone service-create --name=glance \
                      --type=image \
                      --description="Glance Image Service"
keystone service-create --name=keystone \
                      --type=identity \
                      --description="Keystone Identity Service"
keystone service-create --name=swift \
                      --type=object-store \
                      --description="Swift Service"
```

## Setting Up Middleware

### Keystone Auth-Token Middleware

The Keystone auth\_token middleware is a WSGI component that can be inserted in the WSGI pipeline to handle authenticating tokens with Keystone.

### Configuring Nova to use Keystone

When configuring Nova, it is important to create a nova user in the service tenant and include the nova user's login information in /etc/nova/nova.conf

### Configuring Swift to use Keystone

Similar to Nova, swift can be configured to use Keystone for authentication rather than its built in 'tempauth'.

1. Add a service endpoint for Swift to Keystone
2. Configure the paste file for swift-proxy, /etc/swift/proxy-server.conf.
3. Reconfigure Swift's proxy server to use Keystone instead of TempAuth. Here's an example `/etc/swift/proxy-server.conf` :

```
[DEFAULT]
bind_port = 8888
user = <user>

[pipeline:main]
pipeline = catch_errors healthcheck cache auth_token keystone proxy-server

[app:proxy-server]
use = egg:swift#proxy
account_autocreate = true

[filter:keystone]
paste.filter_factory = keystoneclient.middleware.swift_auth:filter_factory
operator_roles = admin, swiftoperator

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

```
# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = 10
service_port = 5000
service_host = 127.0.0.1
auth_port = 35357
auth_host = 127.0.0.1
auth_token = ADMIN
admin_token = ADMIN
cache = swift.cache

[filter:cache]
use = egg:swift#memcache
set log_name = cache

[filter:catch_errors]
use = egg:swift#catch_errors

[filter:healthcheck]
use = egg:swift#healthcheck
```

4. Restart swift services.
5. Verify that the Identity service, Keystone, is providing authentication to Object Storage (Swift).

```
$ swift -v 2 -A http://localhost:5000/v2.0 -U admin:admin -K
ADMIN stat
```

## Configuring Swift with S3 emulation to use Keystone

Keystone support validating S3 tokens using the same tokens as the generated EC2 tokens. When you have generated a pair of EC2 access token and secret you can access your swift cluster directly with the S3 API.

1. Configure the paste file for swift-proxy (`/etc/swift/proxy-server.conf`) to use S3token and Swift3 middleware. You must have the s3token middleware in the pipeline when using keystone and swift3.

Here's an example:

```
[DEFAULT]
bind_port = 8080
user = <user>

[pipeline:main]
pipeline = catch_errors healthcheck cache swift3 s3token auth_token keystone
proxy-server

[app:proxy-server]
use = egg:swift#proxy
account_autocreate = true

[filter:catch_errors]
use = egg:swift#catch_errors
```

```
[filter:healthcheck]
use = egg:swift#healthcheck

[filter:cache]
use = egg:swift#memcache

[filter:swift3]
use = egg:swift#swift3

[filter:keystone]
paste.filter_factory = keystoneclient.middleware.swift_auth:filter_factory
operator_roles = admin, swiftoperator

[filter:s3token]
paste.filter_factory = keystoneclient.middleware.s3_token:filter_factory
auth_port = 35357
auth_host = 127.0.0.1
auth_protocol = http

[filter:authtoken]
paste.filter_factory = keystoneclient.middleware.auth_token:filter_factory
service_port = 5000
service_host = 127.0.0.1
auth_port = 35357
auth_host = 127.0.0.1
auth_protocol = http
auth_token = ADMIN
admin_token = ADMIN
```

2. You can then access directly your Swift via the S3 API, here's an example with the `boto` library:

```
import boto
import boto.s3.connection

connection = boto.connect_s3(
    aws_access_key_id='<ec2 access key for user>',
    aws_secret_access_key='<ec2 secret access key for user>',
    port=8080,
    host='localhost',
    is_secure=False,
    calling_format=boto.s3.connection.OrdinaryCallingFormat())
```

## Configuring Keystone for an LDAP backend

As an alternative to the SQL Database backing store, Keystone can use a directory server to provide the Identity service. An example Schema for OpenStack would look like this:

```
dn: cn=openstack,cn=org
dc: openstack
objectClass: dcObject
objectClass: organizationalUnit
ou: openstack

dn: ou=Groups,cn=openstack,cn=org
objectClass: top
objectClass: organizationalUnit
ou: groups
```

```
dn: ou=Users,cn=openstack,cn=org
objectClass: top
objectClass: organizationalUnit
ou: users

dn: ou=Roles,cn=openstack,cn=org
objectClass: top
objectClass: organizationalUnit
ou: roles
```

The corresponding entries in the Keystone configuration file are:

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

user_tree_dn = ou=Users,dc=openstack,dc=com
user_objectclass = inetOrgPerson

tenant_tree_dn = ou=Groups,dc=openstack,dc=com
tenant_objectclass = groupOfNames

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

The default object classes and attributes are intentionally simplistic. They reflect the common standard objects according to the LDAP RFCs. However, in a live deployment, the correct attributes can be overridden to support a preexisting, more complex schema. For example, in the user object, the objectClass posixAccount from RFC2307 is very common. If this is the underlying objectclass, then the *uid* field should probably be *uidNumber* and *username* field either *uid* or *cn*. To change these two fields, the corresponding entries in the Keystone configuration file are:

```
[ldap]
user_id_attribute = uidNumber
user_name_attribute = cn
```

There is a set of allowed actions per object type that you can modify depending on your specific deployment. For example, the users are managed by another tool and you have only read access, in such case the configuration is:

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

tenant_allow_create = True
tenant_allow_update = True
tenant_allow_delete = True

role_allow_create = True
```

```
role_allow_update = True
role_allow_delete = True
```

There are some configuration options for filtering users, tenants and roles, if the backend is providing too much output, in such case the configuration will look like:

```
[ldap]
user_filter = (memberof=CN=openstack-users,OU=workgroups,DC=openstack,DC=com)
tenant_filter =
role_filter =
```

In case that the directory server does not have an attribute enabled of type boolean for the user, there are several configuration parameters that can be used to extract the value from an integer attribute like in Active Directory:

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

In this case the attribute is an integer and the enabled attribute is listed in bit 1, so the if the mask configured *user\_enabled\_mask* is different from 0, it gets the value from the field *user\_enabled\_attribute* and it makes an ADD operation with the value indicated on *user\_enabled\_mask* and if the value matches the mask then the account is disabled.

It also saves the value without mask to the user identity in the attribute *enabled\_nomask*. This is needed in order to set it back in case that we need to change it to enable/disable a user because it contains more information than the status like password expiration. Last setting *user\_enabled\_mask* is needed in order to create a default value on the integer attribute (512 = NORMAL ACCOUNT on AD)

In case of Active Directory the classes and attributes could not match the specified classes in the LDAP module so you can configure them like so:

```
[ldap]
user_objectclass          = person
user_id_attribute         = cn
user_name_attribute       = cn
user_mail_attribute       = mail
user_enabled_attribute    = userAccountControl
user_enabled_mask         = 2
user_enabled_default      = 512
user_attribute_ignore     = tenant_id,tenants
tenant_objectclass        = groupOfNames
tenant_id_attribute       = cn
tenant_member_attribute   = member
tenant_name_attribute     = ou
tenant_desc_attribute     = description
tenant_enabled_attribute   = extensionName
tenant_attribute_ignore   =
role_objectclass          = organizationalRole
role_id_attribute         = cn
role_name_attribute       = ou
role_member_attribute     = roleOccupant
role_attribute_ignore     =
```

## Reference for LDAP Configuration Options in keystone.conf

**Table 6.1. Description of keystone.conf file configuration options for LDAP**

Configuration option=Default value	(Type) Description
url=ldap://localhost	The location for the ldap server.
user = dc=Manager,dc=example,dc=com	(StrOpt) User for the LDAP server to use as default.
password = None	(StrOpt) Password for LDAP server to connect to.
suffix = cn=example,cn=com	(StrOpt) Default suffix for your LDAP server.
use_dumb_member = False	(Bool) Indicates whether dumb_member settings are in use.
allow_subtree_delete = False	(Bool) Determine whether to delete LDAP subtrees.
dumb_member = cn=dumb,dc=example,dc=com	Mockup member as placeholder, for testing purposes.
query_scope = one	The LDAP scope for queries, this can be either 'one' (onelevel/singleLevel) or 'sub' (subtree/wholeSubtree)
user_tree_dn = ou=Users,dc=example,dc=com	
user_filter =	
user_objectclass = inetOrgPerson	
user_id_attribute = cn	
user_name_attribute = sn	
user_mail_attribute = email	
user_pass_attribute = userPassword	
user_enabled_attribute = enabled	Example, userAccountControl. Combines with user_enabled_mask and user_enabled_default settings below to extract the value from an integer attribute like in Active Directory.
user_enabled_mask = 0	
user_enabled_default = True	
user_attribute_ignore = tenant_id,tenants	
user_allow_create = True	If the users are managed by another tool and you have only read access, you would set this to False.
user_allow_update = True	
user_allow_delete = True	
tenant_tree_dn = ou=Groups,dc=example,dc=com	
tenant_filter =	If the backend is providing too much output, you can set a filter to blank so tenants are not passed through.
tenant_objectclass = groupOfNames	
tenant_id_attribute = cn	
tenant_member_attribute = member	
tenant_name_attribute = ou	
tenant_desc_attribute = desc	
tenant_enabled_attribute = enabled	
tenant_attribute_ignore =	
tenant_allow_create = True	
tenant_allow_update = True	
tenant_allow_delete = True	
role_tree_dn = ou=Roles,dc=example,dc=com	
role_filter =	

Configuration option=Default value	(Type) Description
role_objectclass = organizationalRole	
role_id_attribute = cn	
role_name_attribute = ou	
role_member_attribute = roleOccupant	
role_attribute_ignore =	
role_allow_create = True	
role_allow_update = True	
role_allow_delete = True	
group_tree_dn =	
group_filter =	
group_objectclass = groupOfNames	
group_id_attribute = cn	
group_name_attribute = ou	
group_member_attribute = member	
group_desc_attribute = desc	
group_attribute_ignore =	
group_allow_create = True	
group_allow_update = True	
group_allow_delete = True	

## Auth-Token Middleware with Username and Password

It is also possible to configure Keystone's auth\_token middleware using the 'admin\_user' and 'admin\_password' options. When using the 'admin\_user' and 'admin\_password' options the 'admin\_token' parameter is optional. If 'admin\_token' is specified it will be used only if the specified token is still valid.

Here is an example paste config filter that makes use of the 'admin\_user' and 'admin\_password' parameters:

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

It should be noted that when using this option an admin tenant/role relationship is required. The admin user is granted access to the 'Admin' role on the 'admin' tenant.

## Configuring Keystone SSL support

Keystone may be configured to support 2-way SSL out-of-the-box. The x509 certificates used by Keystone must be obtained externally and configured for use with Keystone as described in this section. However, a set of sample certificates is provided in the

examples/ssl directory with the Keystone distribution for testing. Here is the description of each of them and their purpose:

- ca.pem - Certificate Authority chain to validate against.
- keystone.pem - Public certificate for Keystone server.
- middleware.pem - Public and private certificate for Keystone middleware/client.
- cakey.pem - Private key for the CA.
- keystonekey.pem - Private key for the Keystone server.

Note that you may choose whatever names you want for these certificates, or combine the public/private keys in the same file if you wish. These certificates are just provided as an example.

To enable SSL with client authentication, modify the etc/keystone.conf file accordingly under the [ssl] section. SSL configuration example using the included sample certificates:

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

Example:

```
[ssl]
enable = True
certfile = /home/agent1/openstack/tests/certs/signing_cert.pem
keyfile = /home/agent1/openstack/tests/certs/private_key.pem
ca_certs = /home/agent1/openstack/tests/certs/cacert.pem
cert_required = True
```

- enable: True enables SSL. Defaults to False.
- certfile: Path to Keystone public certificate file.
- keyfile: Path to Keystone private certificate file. If the private key is included in the certfile, the keyfile may be omitted.
- ca\_certs: Path to CA trust chain.
- cert\_required: Requires client certificate. Defaults to False.

## Using External Authentication with OpenStack Identity

When Keystone is executed in apache-httplibd it is possible to use external authentication methods different from the authentication provided by the identity store backend. For example, this makes possible to use a SQL identity backend together with X.509 authentication, Kerberos, etc. instead of using the username/password combination.

## Using HTTPD authentication

Webservers like Apache HTTP support many methods of authentication. Keystone can profit from this feature and let the authentication be done in the webserver, that will pass down the authenticated user to Keystone using the REMOTE\_USER environment variable. This user must exist in advance in the identity backend so as to get a token from the controller. To use this method, OpenStack Identity should be running on apache-  
httpd.

## Using X.509

The following snippet for the Apache conf will authenticate 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>
```

## Troubleshooting Identity (Keystone)

If you see an error opening the signing key file, it's possible the person who ran **keystone-manage pki\_setup** to generate certificates and keys isn't using the correct user. When you run **keystone-manage pki\_setup**, keystone generates a set of certificates and keys in `/etc/keystone/ssl*` which is owned by root:root.

This is problematic when trying to then run the Keystone daemon under the 'keystone' user account (nologin) when trying to run PKI. Unless you manually chown the files `keystone:keystone` or run **keystone-manage pki\_setup** with `--keystone-user` and `--keystone-group` you'll get an error like this:

```
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
```

# 7. Image Management

## Table of Contents

Configuring Tenant-specific Storage Locations for Images with Object Storage .....	116
Adding images with glance image-create .....	116
Getting virtual machine images .....	121
Tool support for creating images .....	122
Customizing an image for OpenStack .....	123
Creating raw or QCOW2 images .....	124
Booting a test image .....	129
Tearing down (deleting) Instances .....	131
Pausing and Suspending Instances .....	131
Select a specific host to boot instances on .....	132
Creating Custom Images .....	133
Creating a Windows Image .....	138
Creating images from running instances with KVM and Xen .....	139
Replicating images across multiple data centers .....	141

You can use OpenStack Image Services for discovering, registering, and retrieving virtual machine images. The service includes a RESTful API that allows users to query VM image metadata and retrieve the actual image with HTTP requests, or you can use a client class in your Python code to accomplish the same tasks.

VM images made available through OpenStack Image Service can be stored in a variety of locations from simple file systems to object-storage systems like the OpenStack Object Storage project, or even use S3 storage either on its own or through an OpenStack Object Storage S3 interface.

The backend stores that OpenStack Image Service can work with are as follows:

- OpenStack Object Storage - OpenStack Object Storage is the highly-available object storage project in OpenStack.
- Filesystem - The default backend that OpenStack Image Service uses to store virtual machine images is the filesystem backend. This simple backend writes image files to the local filesystem.
- S3 - This backend allows OpenStack Image Service to store virtual machine images in Amazon's S3 service.
- HTTP - OpenStack Image Service can read virtual machine images that are available via HTTP somewhere on the Internet. This store is readonly.

This chapter assumes you have a working installation of the Image Service, with a working endpoint and users created in the Identity service, plus you have sourced the environment variables required by the nova client and glance client.

# Configuring Tenant-specific Storage Locations for Images with Object Storage

For some deployers, storing all images in a single place for all tenants and users to access is not ideal. To enable access control to specific images for cloud users, you can configure the Image service with the ability to store image data in the image owner-specific locations.

These are the relevant configuration options in the `glance-api.conf` file:

- `swift_store_multi_tenant`: this must be set to 'True' to enable tenant-specific storage locations (it defaults to 'False').
- `swift_store_admin_tenants`: this is a list of tenants, referenced by id, that should be granted read and write access to all Object Storage containers created by the Image service.

Assuming you configured 'swift' as your `default_store` in `glance-api.conf` and you enable this feature as described above, images will be stored in an Object Storage service (swift) endpoint pulled from the authenticated user's `service_catalog`. The created image data will only be accessible through the Image service by the tenant that owns it and any tenants defined in `swift_store_admin_tenants` that are identified as having admin-level accounts.

## Adding images with glance image-create

Use the `glance image-create` command to add a new virtual machine image to glance, and use `glance image-update` to modify properties of an image that has been updated. The `image-create` command takes several optional arguments, but you should specify a name for your image using the `--name` flag, as well as the disk format with `--disk-format` and container format with `--container-format`. Pass in the file via standard input or using the `file` command. For example:

```
$ glance image-create --name myimage --disk-format=raw --container-format=bare
< /path/to/file.img
```

or

```
$ glance image-create --name myimage --disk-format=raw --container-format=bare
--file /path/to/file.img
```

## Disk and Container Formats for Images

When adding an image to the Image service (glance), you may specify what the virtual machine image's disk format and container format are.

This document explains exactly what these formats are.

### Disk Format

The disk format of a virtual machine image is the format of the underlying disk image. Virtual appliance vendors have different formats for laying out the information contained in a virtual machine disk image.

You can set your image's disk format to one of the following:

- raw

This is an unstructured disk image format; if you have a file without an extension it is possibly a raw format

- vhd

This is the VHD disk format, a common disk format used by virtual machine monitors from VMWare, Xen, Microsoft, VirtualBox, and others

- vmdk

Another common disk format supported by many common virtual machine monitors

- vdi

A disk format supported by VirtualBox virtual machine monitor and the QEMU emulator

- iso

An archive format for the data contents of an optical disc (e.g. CDROM)

- qcow2

A disk format supported by the QEMU emulator that can expand dynamically and supports Copy on Write

- aki

This indicates what is stored in Glance is an Amazon kernel image

- ari

This indicates what is stored in Glance is an Amazon ramdisk image

- ami

This indicates what is stored in Glance is an Amazon machine image

## Container Format

The container format refers to whether the virtual machine image is in a file format that also contains metadata about the actual virtual machine.

Note that the container format string is not currently used by Glance or other OpenStack components, so it is safe to simply specify bare as the container format if you are unsure.

You can set your image's container format to one of the following:

- bare

This indicates there is no container or metadata envelope for the image

- ovf

This is the OVF container format

- aki

This indicates what is stored in Glance is an Amazon kernel image

- ari

This indicates what is stored in Glance is an Amazon ramdisk image

- ami

This indicates what is stored in Glance is an Amazon machine image

## Image metadata

You can associate metadata with an image using the `--property key=value` argument to `glance image-create` or `glance image-update`. For example:

```
$ glance image-update img-uuid --property architecture=arm --property hypervisor_type=qemu
```

If the following properties are set on an image, and the `ImagePropertiesFilter` scheduler filter is enabled (which it is by default), then the scheduler will only consider compute hosts that satisfy these properties:

architecture	The CPU architecture that must be supported by the hypervisor, e.g. <code>x86_64</code> , <code>arm</code> , <code>ppc64</code> . Run <code>uname -m</code> to get the architecture of a machine. We strongly recommend using the architecture data vocabulary defined by the <a href="#">libosinfo project</a> for this purpose. Recognized values for this field are:
alpha	<a href="#">DEC 64-bit RISC</a>
armv7l	<a href="#">ARM Cortex-A7 MPCore</a>
cris	<a href="#">Ethernet, Token Ring, AXIs - Code Reduced Instruction Set</a>
i686	<a href="#">Intel sixth-generation x86 (P6 microarchitecture)</a>
ia64	<a href="#">Itanium</a>
lm32	<a href="#">Lattice Micro32</a>
m68k	<a href="#">Motorola 68000</a>
microblaze	<a href="#">Xilinx 32-bit FPGA (Big Endian)</a>
microblazeel	<a href="#">Xilinx 32-bit FPGA (Little Endian)</a>
mips	<a href="#">MIPS 32-bit RISC (Big Endian)</a>
mipsel	<a href="#">MIPS 32-bit RISC (Little Endian)</a>
mips64	<a href="#">MIPS 64-bit RISC (Big Endian)</a>
mips64el	<a href="#">MIPS 64-bit RISC (Little Endian)</a>
openrisc	<a href="#">OpenCores RISC</a>
parisc	<a href="#">HP Precision Architecture RISC</a>
parisc64	<a href="#">HP Precision Architecture 64-bit RISC</a>
ppc	<a href="#">PowerPC 32-bit</a>
ppc64	<a href="#">PowerPC 64-bit</a>
ppcemb	<a href="#">PowerPC (Embedded 32-bit)</a>
s390	<a href="#">IBM Enterprise Systems Architecture/390</a>
s390x	<a href="#">S/390 64-bit</a>

sh4	SuperH SH-4 (Little Endian)
sh4eb	SuperH SH-4 (Big Endian)
sparc	Scalable Processor Architecture, 32-bit
sparc64	Scalable Processor Architecture, 64-bit
unicore32	Microprocessor Research and Development Center RISC Unicore32
x86_64	64-bit extension of IA-32
xtensa	Tensilica Xtensa configurable microprocessor core
xtensaeb	Tensilica Xtensa configurable microprocessor core (Big Endian)
hypervisor_type	The hypervisor type. Allowed values include: xen, qemu, kvm, lxc, uml, vmware, hyperv, powervm.
vm_mode	The virtual machine mode. This represents the host/guest ABI (application binary interface) used for the virtual machine. Allowed values are:  hvm Fully virtualized. This is the mode used by QEMU and KVM.  xen Xen 3.0 paravirtualized.  uml User Mode Linux paravirtualized.  exe Executables in containers. This is the mode used by LXC.

The following metadata properties are specific to the XenAPI driver:

auto_disk_config	A boolean option. If true, the root partition on the disk will be automatically resized before the instance boots. This value is only taken into account by the Compute service when using a Xen-based hypervisor with the XenAPI driver. The Compute service will only attempt to resize if there is a single partition on the image, and only if the partition is in ext3 or ext4 format.
os_type	The operating system installed on the image, e.g. linux, windows. The XenAPI driver contains logic that will take different actions depending on the value of the os_type parameter of the image. For example, for images where os_type=windows, it will create a FAT32-based swap partition instead of a Linux swap partition, and it will limit the injected hostname to less than 16 characters.

The following metadata properties are specific to the VMware API driver:

vmware_adaptertype	Indicates the virtual SCSI or IDE controller used by the hypervisor. Allowed values: lsiLogic, busLogic, ide
vmware_ostype	A VMware GuestID which describes the operating system installed in the image. This will be passed to the hypervisor when creating a virtual machine. See <a href="#">thinkvirt.com</a> for a list of valid values. If this is not specified, it will default to otherGuest.
vmware_image_version	Currently unused, set it to 1.

In order to assist end-users in utilizing images, you may wish to put additional common metadata on Glance images. By community agreement, the following metadata keys may be used across Glance installations for the purposes described below.

instance_uuid	For snapshot images, this is the UUID of the server used to create this image.
kernel_id	The ID of image stored in Glance that should be used as the kernel when booting an AMI-style image.
ramdisk_id	The ID of image stored in Glance that should be used as the ramdisk when booting an AMI-style image.
os_version	The operating system version as specified by the distributor.
os_distro	The value of this property is the common name of the operating system distribution in all-lowercase. For this purpose, we use the same data vocabulary as the <a href="#">libosinfo project</a> . Following are the recognized values for this property. In the interest of interoperability, please use only a recognized value for this field. The deprecated values are listed to assist you in searching for the recognized value. Allowed values are: <ul style="list-style-type: none"><li>arch This is: Arch Linux Do not use: archlinux, or org.archlinux</li><li>centos This is: Community Enterprise Operating System Do not use: org.centos CentOS</li><li>debian This is: Debian Do not use: Debian, or org.debian</li><li>fedora This is: Fedora Do not use: Fedora, org.fedora, or org.fedoraproject</li><li>freebsd This is: FreeBSD Do not use: org.freebsd, freeBSD, or FreeBSD</li><li>gentoo This is: Gentoo Linux Do not use: Gentoo, or org.gentoo</li><li>mandrake This is: Mandrakelinux (MandrakeSoft) Do not use: mandrakelinux, or MandrakeLinux</li><li>mandriva This is: Mandriva Linux Do not use: mandrivalinux</li></ul>

mes	This is: Mandriva Enterprise Server  Do not use: mandrivaent, or mandrivaES
msdos	This is: Microsoft Disc Operating System  Do not use: ms-dos
netbsd	This is: NetBSD  Do not use: NetBSD, or org.netbsd
netware	This is: Novell NetWare  Do not use: novell, or NetWare
openbsd	This is: OpenBSD  Do not use: OpenBSD, or org.openbsd
opensolaris	Do not use: OpenSolaris, or org.opensolaris
opensuse	This is: openSUSE  Do not use: suse, SuSE, or org.opensuse
rhel	This is: Red Hat Enterprise Linux  Do not use: redhat, RedHat, or com.redhat
sled	This is: SUSE Linux Enterprise Desktop  Do not use: com.suse
ubuntu	This is: Ubuntu  Do not use: Ubuntu, com.ubuntu, org.ubuntu, or canonical
windows	This is: Microsoft Windows  Do not use: com.microsoft.server, or windoze

## Getting virtual machine images

### CirrOS (test) images

Scott Moser maintains a set of small virtual machine images that are designed for testing. These images use `cirros` as the login user. They are hosted under the CirrOS project on Launchpad and are available for download.

---

If your deployment uses QEMU or KVM, we recommend using the images in QCOW2 format. The most recent 64-bit QCOW2 image as of this writing is [cirros-0.3.0-x86\\_64-disk.img](#).

## Ubuntu images

Canonical maintains an [official set of Ubuntu-based images](#). These accounts use `ubuntu` as the login user.

If your deployment uses QEMU or KVM, we recommend using the images in QCOW2 format. The most recent version of the 64-bit QCOW2 image for Ubuntu 12.04 is [precise-server-cloudimg-amd64-disk1.img](#).

## Fedora images

The Fedora project maintains prebuilt Fedora JEOS (Just Enough OS) images for download at <http://berrange.fedorapeople.org/images>.

A 64-bit QCOW2 image for Fedora 16, [f16-x86\\_64-openstack-sda.qcow2](#), is available for download.

## openSUSE and SLES 11 images

[SUSE Studio](#) is an easy way to build virtual appliances for openSUSE and SLES 11 (SUSE Linux Enterprise Server) that are compatible with OpenStack. Free registration is required to download or build images.

For example, Christian Berendt used openSUSE to create a [test openSUSE 12.1 \(JeOS\) image](#).

## Rackspace Cloud Builders (multiple distros) images

Rackspace Cloud Builders maintains a list of pre-built images from various distributions (RedHat, CentOS, Fedora, Ubuntu) at [rackerjoe/oz-image-build](#) on Github.

## Tool support for creating images

There are several open-source third-party tools available that simplify the task of creating new virtual machine images.

### Oz (KVM)

[Oz](#) is a command-line tool that has the ability to create images for common Linux distributions. Rackspace Cloud Builders uses Oz to create virtual machines, see [rackerjoe/oz-image-build](#) on Github for their Oz templates. For an example from the Fedora Project wiki, see [Building an image with Oz](#).

### VMBuilder (KVM, Xen)

[VMBuilder](#) can be used to create virtual machine images for different hypervisors.

The [Ubuntu 12.04 server guide](#) has documentation on how to use VMBuilder.

## BoxGrinder (KVM, Xen, VMWare)

[BoxGrinder](#) is another tool for creating virtual machine images, which it calls appliances. BoxGrinder can create Fedora, Red Hat Enterprise Linux, or CentOS images. BoxGrinder is currently only supported on Fedora.

## VeeWee (KVM)

[VeeWee](#) is often used to build [Vagrant](#) boxes, but it can also be used to build KVM images.

See the [doc/definition.md](#) and [doc/template.md](#) VeeWee documentation files for more details.

## imagefactory

[imagefactory](#) is a new tool from the [Aeolus](#) project designed to automate the building, converting, and uploading images to different cloud providers. It includes support for OpenStack-based clouds.

## Customizing an image for OpenStack

This section describes what customizations you should do to your image to maximize compatibility with OpenStack.

### Support metadata service or config drive

An image needs to be able to retrieve information from OpenStack, such as the ssh public key and [user data](#) that the user submitted when requesting the image. This information is accessible via the metadata service or the config drive. The easiest way to support this is to install the [cloud-init](#) package into your image.

### Support resizing

The size of the disk in a virtual machine image is determined when you initially create the image. However, OpenStack lets you launch instances with different size drives by specifying different flavors. For example, if your image was created with a 5 GB disk, and you launch an instance with a flavor of `m1.small`, the resulting virtual machine instance will have a primary disk of 10GB. When an instance's disk is resized up, zeros are just added to the end.

Your image needs to be able to resize its partitions on boot to match the size requested by the user. Otherwise, after the instance boots, you will need to manually resize the partitions if you want to access the additional storage you have access to when the disk size associated with the flavor exceeds the disk size your image was created with.

Your image must be configured to deal with two issues:

- The image's partition table describes the original size of the image
- The image's filesystem fills the original size of the image

## Adjusting the partition table on instance boot

Your image will need to run a script on boot to modify the partition table. Due to a limitation in the Linux kernel, you cannot modify a partition table of a disk that has partition currently mounted (you can for LVM, but not for "raw disks"); this partition adjustment has to happen inside the initramfs before the root volume is mounted, or a reboot has to be done to free the mount of `/`.

Ubuntu cloud images and cirros images use a tool called **growpart** that is part of the [cloud-utils](#) package.

## Adjusting the filesystem

You will need to resize the file system in addition to the partition table. If you have cloud-init installed, it will do the resize assuming the partition tables have been adjusted properly. Cirros images run `resize2fs` on the root partition on boot.



### Note

If you are using XenServer as your hypervisor, the above steps are not needed as the Compute service will automatically adjust the partition and filesystem for your instance on boot. Automatic resize will occur if the following are all true:

- `auto_disk_config=True` in `nova.conf`.
- The disk on the image has only one partition.
- The file system on the one partition is ext3 or ext4.

## Creating raw or QCOW2 images

This section describes how to create a raw or QCOW2 image from a Linux installation ISO file. Raw images are the simplest image file format and are supported by all of the hypervisors. QCOW2 images have several advantages over raw images. They take up less space than raw images (growing in size as needed), and they support snapshots.



### Note

QCOW2 images are only supported with KVM and QEMU hypervisors.

As an example, this section will describe how to create a CentOS 6.2 image. [64-bit ISO images of CentOS 6.2](#) can be downloaded from one of the CentOS mirrors. This example uses the CentOS netinstall ISO, which is a smaller ISO file that downloads packages from the Internet as needed.

## Create an empty image (raw)

Here we create a 5GB raw image using the `kvm-img` command:

```
$ IMAGE=centos-6.2.img
$ kvm-img create -f raw $IMAGE 5G
```

## Create an empty image (QCOW2)

Here we create a a 5GB QCOW2 image using the **kvm-img** command:

```
$ IMAGE=centos-6.2.img
$ kvm-img create -f qcow $IMAGE 5G
```

## Boot the ISO using the image

First, find a spare vnc display. (Note that vnc display :N correspond to TCP port 5900+N, so that :0 corresponds to port 5900). Check which ones are currently in use with the **lsof** command, as root:

```
# lsof -i | grep "TCP *:590"
kvm      3437 libvirt-qemu    14u  IPv4 1629164      0t0  TCP *:5900 (LISTEN)
kvm      24966 libvirt-qemu    24u  IPv4 1915470      0t0  TCP *:5901
(LISTEN)
```

This shows that vnc displays :0 and :1 are in use. In this example, we will use VNC display :2.

Also, we want a temporary file to send power signals to the VM instance. We default to /tmp/file.mon, but make sure it doesn't exist yet. If it does, use a different file name for the MONITOR variable defined below:

```
$ IMAGE=centos-6.2.img
$ ISO=CentOS-6.2-x86_64-netinstall.iso
$ VNCDISPLAY=:2
$ MONITOR=/tmp/file.mon
$ sudo kvm -m 1024 -cdrom $ISO -drive file=${IMAGE},if=virtio,index=0 \
-boot d -net nic -net user -nographic -vnc ${VNCDISPLAY} \
-monitor unix:${MONITOR},server,nowait
```

## Connect to the instance via VNC

VNC is a remote desktop protocol that will give you full-screen display access to the virtual machine instance, as well as let you interact with keyboard and mouse. Use a VNC client (e.g., [Vinagre](#) on Gnome, [Krdc](#) on KDE, xvnc4viewer from [RealVNC](#), xtightvncviewer from [TightVNC](#)) to connect to the machine using the display you specified. You should now see a CentOS install screen.

## Point the installer to a CentOS web server

The CentOS net installer requires that the user specify the web site and a CentOS directory that corresponds to one of the CentOS mirrors.

- Web site name: `mirror.umd.edu` (consider using other mirrors as an alternative)
- CentOS directory: `centos/6.2/os/x86_64`

See [CentOS mirror page](#) to get a full list of mirrors, click on the "HTTP" link of a mirror to retrieve the web site name of a mirror.

## Partition the disks

There are different options for partitioning the disks. The default installation will use LVM partitions, and will create three partitions (/boot, /, swap). The simplest approach is to create a single ext4 partition, mounted to "/".

## Step through the install

The simplest thing to do is to choose the "Server" install, which will install an SSH server.

## When install completes, shut down the instance

Power down the instance using the monitor socket file to send a power down signal, as root:

```
# MONITOR=/tmp/file.mon
# echo 'system_powerdown' | socat - UNIX-CONNECT:$MONITOR
```

## Start the instance again without the ISO

```
$ VNCDISPLAY=:2
$ MONITOR=/tmp/file.mon
$ sudo kvm -m 1024 -drive file=${IMAGE},if=virtio,index=0 \
-boot c -net nic -net user -nographic -vnc ${VNCDISPLAY} \
-monitor unix:${MONITOR},server,nowait
```

## Connect to instance via VNC

When you boot the first time, it will ask you about authentication tools, you can just choose 'Exit'. Then, log in as root using the root password you specified.

## Edit HWADDR from eth0 config file

The operating system records the MAC address of the virtual ethernet card in /etc/sysconfig/network-scripts/ifcfg-eth0 during the instance process. However, each time the image boots up, the virtual ethernet card will have a different MAC address, so this information must be deleted from the configuration file.

Edit /etc/sysconfig/network-scripts/ifcfg-eth0 and remove the HWADDR= line.

## Configure to fetch metadata

An instance must perform several steps on startup by interacting with the metadata service (e.g., retrieve ssh public key, execute user data script). There are several ways to implement this functionality, including:

- Install a [cloud-init RPM](#), which is a port of the Ubuntu [cloud-init](#) package. This is the recommended approach.
- Modify `/etc/rc.local` to fetch desired information from the metadata service, as described below.

## Using cloud-init to fetch the public key

The cloud-init package will automatically fetch the public key from the metadata server and place the key in an account. The account varies by distribution. On Ubuntu-based virtual machines, the account is called "ubuntu". On Fedora-based virtual machines, the account is called "ec2-user".

You can change the name of the account used by cloud-init by editing the `/etc/cloud/cloud.cfg` file and adding a line with a different user. For example, to configure cloud-init to put the key in an account named "admin", edit the config file so it has the line:

```
user: admin
```

## Writing a script to fetch the public key

To fetch the ssh public key and add it to the root account, edit the `/etc/rc.local` file and add the following lines before the line "touch `/var/lock/subsys/local`"

```
depmod -a
modprobe acpiphp

# simple attempt to get the user ssh key using the meta-data service
mkdir -p /root/.ssh
echo >> /root/.ssh/authorized_keys
curl -m 10 -s http://169.254.169.254/latest/meta-data/public-keys/0/openssh-key | grep 'ssh-rsa' >> /root/.ssh/authorized_keys
echo "AUTHORIZED_KEYS:"
echo "*****"
cat /root/.ssh/authorized_keys
echo "*****"
```



### Note

Some VNC clients replace : (colon) with ; (semicolon) and \_ (underscore) with - (hyphen). Make sure it's http: not http; and authorized\_keys not authorized-keys.



### Note

The above script only retrieves the ssh public key from the metadata server. It does not retrieve *user data*, which is optional data that can be passed by the user when requesting a new instance. User data is often used for running a custom script when an instance comes up.

As the OpenStack metadata service is compatible with version 2009-04-04 of the Amazon EC2 metadata service, consult the Amazon EC2 documentation on [Using Instance Metadata](#) for details on how to retrieve user data.

## Shut down the instance

From inside the instance, as root:

```
# /sbin/shutdown -h now
```

## Modifying the image (raw)

You can make changes to the filesystem of an image without booting it, by mounting the image as a file system. To mount a raw image, you need to attach it to a loop device (e.g., `/dev/loop0`, `/dev/loop1`). To identify the next unused loop device, as root:

```
# losetup -f  
/dev/loop0
```

In the example above, `/dev/loop0` is available for use. Associate it to the image using `losetup`, and expose the partitions as device files using `kpartx`, as root:

```
# IMAGE=centos-6.2.img  
# losetup /dev/loop0 $IMAGE  
# kpartx -av /dev/loop0
```

If the image has, say three partitions (`/boot`, `/`, `/swap`), there should be one new device created per partition:

```
$ ls -l /dev/mapper/loop0p*  
brw-rw---- 1 root disk 43, 49 2012-03-05 15:32 /dev/mapper/loop0p1  
brw-rw---- 1 root disk 43, 50 2012-03-05 15:32 /dev/mapper/loop0p2  
brw-rw---- 1 root disk 43, 51 2012-03-05 15:32 /dev/mapper/loop0p3
```

To mount the second partition, as root:

```
# mkdir /mnt/image  
# mount /dev/mapper/loop0p2 /mnt/image
```

You can now modify the files in the image by going to `/mnt/image`. When done, unmount the image and release the loop device, as root:

```
# umount /mnt/image  
# losetup -d /dev/loop0
```

## Modifying the image (qcow2)

You can make changes to the filesystem of an image without booting it, by mounting the image as a file system. To mount a QEMU image, you need the nbd kernel module to be loaded. Load the nbd kernel module, as root:

```
# modprobe nbd max_part=8
```



### Note

If nbd has already been loaded with `max_part=0`, you will not be able to mount an image if it has multiple partitions. In this case, you may need to first unload the nbd kernel module, and then load it. To unload it, as root:

```
# rmmod nbd
```

Connect your image to one of the network block devices (e.g., `/dev/nbd0`, `/dev/nbd1`). In this example, we use `/dev/nbd3`. As root:

```
# IMAGE=centos-6.2.img
# qemu-nbd -c /dev/nbd3 $IMAGE
```

If the image has, say three partitions (`/boot`, `/`, `/swap`), there should be one new device created per partition:

```
$ ls -l /dev/nbd3*
brw-rw---- 1 root disk 43, 48 2012-03-05 15:32 /dev/nbd3
brw-rw---- 1 root disk 43, 49 2012-03-05 15:32 /dev/nbd3p1
brw-rw---- 1 root disk 43, 50 2012-03-05 15:32 /dev/nbd3p2
brw-rw---- 1 root disk 43, 51 2012-03-05 15:32 /dev/nbd3p3
```



## Note

If the network block device you selected was already in use, the initial `qemu-nbd` command will fail silently, and the `/dev/nbd3p{1,2,3}` device files will not be created.

To mount the second partition, as root:

```
# mkdir /mnt/image
# mount /dev/nbd3p2 /mnt/image
```

You can now modify the files in the image by going to `/mnt/image`. When done, unmount the image and release the network block device, as root:

```
# umount /mnt/image
# qemu-nbd -d /dev/nbd3
```

## Upload the image to glance (raw)

```
$ IMAGE=centos-6.2.img
$ NAME=centos-6.2
$ glance image-create --name="${NAME}" --is-public=true --container-format=ovf
--disk-format=raw < ${IMAGE}
```

## Upload the image to glance (qcow2)

```
$ IMAGE=centos-6.2.img
$ NAME=centos-6.2
$ glance image-create --name="${NAME}" --is-public=true --container-format=ovf
--disk-format=qcow2 < ${IMAGE}
```

## Booting a test image

The following assumes you are using QEMU or KVM in your deployment.

Download a CirrOS test image:

```
$ wget https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-disk.img
```

Add the image to glance:

```
$ name=cirros-0.3.0-x86_64
$ image=cirros-0.3.0-x86_64-disk.img
$ glance image-create --name=$name --is-public=true --container-format=bare --disk-format=qcow2 < $image
```

Check that adding the image was successful (Status should be ACTIVE when the operation is complete):

```
$ nova image-list
```

ID	Name	Status	Server
254c15e1-78a9-4b30-9b9e-2a39b985001c	cirros-0.3.0-x86_64	ACTIVE	

Create a keypair so you can ssh to the instance:

```
$ nova keypair-add test > test.pem
$ chmod 600 test.pem
```

In general, you need to use an ssh keypair to log in to a running instance, although some images have built-in accounts created with associated passwords. However, since images are often shared by many users, it is not advised to put passwords into the images. Nova therefore supports injecting ssh keys into instances before they are booted. This allows a user to log in to the instances that he or she creates securely. Generally the first thing that a user does when using the system is create a keypair.

Keypairs provide secure authentication to your instances. As part of the first boot of a virtual image, the private key of your keypair is added to authorized\_keys file of the login account. Nova generates a public and private key pair, and sends the private key to the user. The public key is stored so that it can be injected into instances.

Run (boot) a test instance:

```
$ nova boot --image cirros-0.3.0-x86_64 --flavor m1.small --key_name test my-first-server
```

Here's a description of the parameters used above:

- **--image**: the name or ID of the image we want to launch, as shown in the output of **nova image-list**
- **--flavor**: the name or ID of the size of the instance to create (number of vcpus, available RAM, available storage). View the list of available flavors by running **nova flavor-list**
- **-key\_name**: the name of the key to inject into the instance at launch.

Check the status of the instance you launched:

```
$ nova list
```

The instance will go from BUILD to ACTIVE in a short time, and you should be able to connect via ssh as 'cirros' user, using the private key you created. If your ssh keypair fails for some reason, you can also log in with the default cirros password: cubswin: )

```
$ ipaddress=... # Get IP address from "nova list"  
$ ssh -i test.pem -l cirros $ipaddress
```

The 'cirros' user is part of the sudoers group, so you can escalate to 'root' via the following command when logged in to the instance:

```
$ sudo -i
```

## Tearing down (deleting) Instances

When you are done with an instance, you can tear it down using the **nova delete** command, passing either the instance name or instance ID as the argument. You can get a listing of the names and IDs of all running instances using the **nova list**. For example:

```
$ nova list
```

ID	Name	Status	Networks
8a5d719a-b293-4a5e-8709-a89b6ac9cee2	my-first-server	ACTIVE	

```
$ nova delete my-first-server
```

## Pausing and Suspending Instances

Since the release of the API in its 1.1 version, it is possible to pause and suspend instances.



## Warning

Pausing and Suspending instances only apply to KVM-based hypervisors and XenServer/XCP Hypervisors.

Pause/ Unpause : Stores the content of the VM in memory (RAM).

Suspend/ Resume : Stores the content of the VM on disk.

It can be interesting for an administrator to suspend instances, if a maintenance is planned; or if the instance are not frequently used. Suspending an instance frees up memory and vCPU, while pausing keeps the instance running, in a "frozen" state. Suspension could be compared to an "hibernation" mode.

## Pausing instance

To pause an instance :

```
nova pause $server-id
```

To resume a paused instance :

```
nova unpause $server-id
```

## Suspending instance

To suspend an instance :

```
nova suspend $server-id
```

To resume a suspended instance :

```
nova resume $server-id
```

## Select a specific host to boot instances on

If you have the appropriate permissions, you can select the specific host where the instance will be launched. This is done using the `--availability_zone zone:host` arguments to the `nova boot` command. For example:

```
$ nova boot --image <uuid> --flavor m1.tiny --key_name test --availability-zone nova:server2
```

Starting with the Grizzly release, you can specify which roles are permitted to boot an instance to a specific host with the `create:forced_host` setting within `policy.json` on the desired roles. By default, only the admin role has this setting enabled.

You can view the list of valid compute hosts by using the `nova hypervisor-list` command, for example:

```
$ nova hypervisor-list
+-----+
| ID | Hypervisor hostname |
```

1	server2
2	server3
3	server4



### Note

The `--availability_zone zone:host` flag replaced the `--force_hosts` scheduler hint for specifying a specific host, starting with the Folsom release.

## Creating Custom Images

There are several pre-built images for OpenStack available from various sources. You can download such images and use them to get familiar with OpenStack. You can refer to <http://docs.openstack.org/trunk/openstack-compute/admin/content/startng-images.html> for details on using such images.

For any production deployment, you may like to have the ability to bundle custom images, with a custom set of applications or configuration. This chapter will guide you through the process of creating Linux images of Debian and Redhat based distributions from scratch. We have also covered an approach to bundling Windows images.

There are some minor differences in the way you would bundle a Linux image, based on the distribution. Ubuntu makes it very easy by providing cloud-init package, which can be used to take care of the instance configuration at the time of launch. cloud-init handles importing ssh keys for password-less login, setting hostname etc. The instance acquires the instance specific configuration from Nova-compute by connecting to a meta data interface running on 169.254.169.254.

While creating the image of a distro that does not have cloud-init or an equivalent package, you may need to take care of importing the keys etc. by running a set of commands at boot time from rc.local.

The process used for Ubuntu and Fedora is largely the same with a few minor differences, which are explained below.

In both cases, the documentation below assumes that you have a working KVM installation to use for creating the images. We are using the machine called 'client1' as explained in the chapter on "Installation and Configuration" for this purpose.

The approach explained below will give you disk images that represent a disk without any partitions. Nova-compute can resize such disks (including resizing the file system) based on the instance type chosen at the time of launching the instance. These images cannot have 'bootable' flag and hence it is mandatory to have associated kernel and ramdisk images. These kernel and ramdisk images need to be used by nova-compute at the time of launching the instance.

However, we have also added a small section towards the end of the chapter about creating bootable images with multiple partitions that can be used by nova to launch an instance without the need for kernel and ramdisk images. The caveat is that while nova-

compute can re-size such disks at the time of launching the instance, the file system size is not altered and hence, for all practical purposes, such disks are not re-sizable.

## Creating a Linux Image – Ubuntu & Fedora

The first step would be to create a raw image on Client1. This will represent the main HDD of the virtual machine, so make sure to give it as much space as you will need.

```
kvm-img create -f raw server.img 5G
```

### OS Installation

Download the iso file of the Linux distribution you want installed in the image. The instructions below are tested on Ubuntu 11.04 Natty Narwhal 64-bit server and Fedora 14 64-bit. Most of the instructions refer to Ubuntu. The points of difference between Ubuntu and Fedora are mentioned wherever required.

```
wget http://releases.ubuntu.com/natty/ubuntu-11.04-server-amd64.iso
```

Boot a KVM instance with the OS installer ISO in the virtual CD-ROM. This will start the installation process. The command below also sets up a VNC display at port 0

```
sudo kvm -m 256 -cdrom ubuntu-11.04-server-amd64.iso -drive file=server.img,if=scsi,index=0 -boot d -net nic -net user -nographic -vnc :0
```

Connect to the VM through VNC (use display number :0) and finish the installation.

For example, where 10.10.10.4 is the IP address of client1:

```
vncviewer 10.10.10.4 :0
```

During the installation of Ubuntu, create a single ext4 partition mounted on '/'. Do not create a swap partition.

In the case of Fedora 14, the installation will not progress unless you create a swap partition. Please go ahead and create a swap partition.

After finishing the installation, relaunch the VM by executing the following command.

```
sudo kvm -m 256 -drive file=server.img,if=scsi,index=0 -boot c -net nic -net user -nographic -vnc :0
```

At this point, you can add all the packages you want to have installed, update the installation, add users and make any configuration changes you want in your image.

At the minimum, for Ubuntu you may run the following commands

```
sudo apt-get update
sudo apt-get upgrade
sudo apt-get install openssh-server cloud-init
```

For Fedora run the following commands as root

```
yum update
yum install openssh-server
chkconfig sshd on
```

Also remove the network persistence rules from /etc/udev/rules.d as their presence will result in the network interface in the instance coming up as an interface other than eth0.

```
sudo rm -rf /etc/udev/rules.d/70-persistent-net.rules
```

Shut down the Virtual machine and proceed with the next steps.

## Extracting the EXT4 partition

The image that needs to be uploaded to OpenStack needs to be an ext4 filesystem image. Here are the steps to create an ext4 filesystem image from the raw image i.e server.img

```
sudo losetup -f server.img
sudo losetup -a
```

You should see an output like this:

```
/dev/loop0: [0801]:16908388 ($filepath)
```

Observe the name of the loop device ( /dev/loop0 in our setup) when \$filepath is the path to the mounted .raw file.

Now we need to find out the starting sector of the partition. Run:

```
sudo fdisk -cul /dev/loop0
```

You should see an output like this:

```
Disk /dev/loop0: 5368 MB, 5368709120 bytes
149 heads, 8 sectors/track, 8796 cylinders, total 10485760 sectors
Units = sectors of 1 * 512 = 512 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0x00072bd4
Device Boot      Start        End      Blocks   Id  System
/dev/loop0p1    *        2048     10483711     5240832   83  Linux
```

Make a note of the starting sector of the /dev/loop0p1 partition i.e the partition whose ID is 83. This number should be multiplied by 512 to obtain the correct value. In this case: 2048 x 512 = 1048576

Unmount the loop0 device:

```
sudo losetup -d /dev/loop0
```

Now mount only the partition (/dev/loop0p1) of server.img which we had previously noted down, by adding the -o parameter with the previously calculated value

```
sudo losetup -f -o 1048576 server.img
sudo losetup -a
```

You'll see a message like this:

```
/dev/loop0: [0801]:16908388 ($filepath) offset 1048576
```

Make a note of the mount point of our device(/dev/loop0 in our setup) when \$filepath is the path to the mounted .raw file.

Copy the entire partition to a new .raw file

```
sudo dd if=/dev/loop0 of=serverfinal.img
```

Now we have our ext4 filesystem image i.e serverfinal.img

Unmount the loop0 device

```
sudo losetup -d /dev/loop0
```

## Tweaking /etc/fstab

You will need to tweak /etc/fstab to make it suitable for a cloud instance. Nova-compute may resize the disk at the time of launch of instances based on the instance type chosen. This can make the UUID of the disk invalid. Hence we have to use a file system label as the identifier for the partition instead of the UUID.

Loop mount the serverfinal.img, by running

```
sudo mount -o loop serverfinal.img /mnt
```

Edit /mnt/etc/fstab and modify the line for mounting root partition(which may look like the following)

```
UUID=e7f5af8d-5d96-45cc-a0fc-d0d1bde8f31c   /
remount-ro 0          1                         ext4      errors=
```

to

```
LABEL=uec-rootfs   /
ext4      defaults  0      0
```

## Fetching Metadata in Fedora

An instance must perform several steps on startup by interacting with the metadata service (e.g., retrieve ssh public key, execute user data script). When building a Fedora image, there are several options for implementing this functionality, including:

- Install a [cloud-initRPM](#), which is a port of the Ubuntu cloud-init package.
- Install [Condenser](#), an alternate version of cloud-init.
- Modify `/etc/rc.local` to fetch desired information from the metadata service, as described below.

To fetch the ssh public key and add it to the root account, edit the `/etc/rc.local` file and add the following lines before the line “`touch /var/lock/subsys/local`”

```
depmod -a
modprobe acpiphp

# simple attempt to get the user ssh key using the meta-data service
mkdir -p /root/.ssh
echo >> /root/.ssh/authorized_keys
curl -m 10 -s http://169.254.169.254/latest/meta-data/public-keys/0/openssh-
key | grep 'ssh-rsa' >> /root/.ssh/authorized_keys
echo "AUTHORIZED_KEYS:"
echo "*****"
cat /root/.ssh/authorized_keys
echo "*****"
```



### Note

The above script only retrieves the ssh public key from the metadata server. It does not retrieve *user data*, which is optional data that can be passed by the user when requesting a new instance. User data is often used for running a custom script when an instance comes up.

As the OpenStack metadata service is compatible with version 2009-04-04 of the Amazon EC2 metadata service, consult the Amazon EC2 documentation on [Using Instance Metadata](#) for details on how to retrieve user data.

## Kernel and Initrd for OpenStack

Copy the kernel and the initrd image from `/mnt/boot` to user home directory. These will be used later for creating and uploading a complete virtual image to OpenStack.

```
sudo cp /mnt/boot/vmlinuz-2.6.38-7-server /home/localadmin
sudo cp /mnt/boot/initrd.img-2.6.38-7-server /home/localadmin
```

Unmount the Loop partition

```
sudo umount /mnt
```

Change the filesystem label of `serverfinal.img` to ‘`uec-rootfs`’

```
sudo tune2fs -L uec-rootfs serverfinal.img
```

Now, we have all the components of the image ready to be uploaded to OpenStack imaging server.

## Registering with OpenStack

The last step would be to upload the images to OpenStack Image Service. The files that need to be uploaded for the above sample setup of Ubuntu are: vmlinuz-2.6.38-7-server, initrd.img-2.6.38-7-server, serverfinal.img

Run the following command

```
uec-publish-image -t image --kernel-file vmlinuz-2.6.38-7-server --ramdisk-file initrd.img-2.6.38-7-server amd64 serverfinal.img bucket1
```

For Fedora, the process will be similar. Make sure that you use the right kernel and initrd files extracted above.

The uec-publish-image command returns the prompt back immediately. However, the upload process takes some time and the images will be usable only after the process is complete. You can keep checking the status using the command **nova image-list** as mentioned below.

## Bootable Images

You can register bootable disk images without associating kernel and ramdisk images. When you do not want the flexibility of using the same disk image with different kernel/ramdisk images, you can go for bootable disk images. This greatly simplifies the process of bundling and registering the images. However, the caveats mentioned in the introduction to this chapter apply. Please note that the instructions below use server.img and you can skip all the cumbersome steps related to extracting the single ext4 partition.

```
glance image-create --name="My Server" --is-public=true --container-format=ovf --disk-format=raw < server.img
```

## Image Listing

The status of the images that have been uploaded can be viewed by using **nova image-list** command. The output should like this:

```
nova image-list
```

ID	Name	Status
6	ttylinux-uec-amd64-12.1_2.6.35-22_1-vmlinuz	ACTIVE
7	ttylinux-uec-amd64-12.1_2.6.35-22_1-initrd	ACTIVE
8	ttylinux-uec-amd64-12.1_2.6.35-22_1.img	ACTIVE

## Creating a Windows Image

The first step would be to create a raw image on Client1, this will represent the main HDD of the virtual machine, so make sure to give it as much space as you will need.

```
kvm-img create -f raw windowsserver.img 20G
```

OpenStack presents the disk using aVIRTIO interface while launching the instance. Hence the OS needs to have drivers for VIRTIO. By default, the Windows Server 2008 ISO does not have the drivers for VIRTIO. So download a virtual floppy drive containing VIRTIO drivers from the following location

<http://alt.fedoraproject.org/pub/alt/virtio-win/latest/images/bin/>

and attach it during the installation

Start the installation by running

```
sudo kvm -m 2048 -cdrom win2k8_dvd.iso -drive file=windowsserver.img,if=virtio  
-drive file=virtio-win-0.1-22.iso,index=3,media=cdrom -net nic,model=virtio -  
net user -nographic -vnc :0
```

When the installation prompts you to choose a hard disk device you won't see any devices available. Click on "Load drivers" at the bottom left and load the drivers from A:\i386\Win2008

After the Installation is over, boot into it once and install any additional applications you need to install and make any configuration changes you need to make. Also ensure that RDP is enabled as that would be the only way you can connect to a running instance of Windows. Windows firewall needs to be configured to allow incoming ICMP and RDP connections.

For OpenStack to allow incoming RDP Connections, use commands to open up port 3389.

Shut-down the VM and upload the image to OpenStack

```
glance image-create --name="My WinServer" --is-public=true --container-format=  
ovf --disk-format=raw < windowsserver.img
```

## Creating images from running instances with KVM and Xen

It is possible to create an image from a running instance on KVM and Xen. This is a convenient way to spawn pre-configured instances; update them according to your needs ; and re-image the instances. The process to create an image from a running instance is quite simple :

- **Pre-requisites (KVM)**

In order to use the feature properly, you will need **qemu-img** 0.14 or greater. The imaging feature uses the copy from a snapshot for image files. (e.g **qcow-img convert -f qcow2 -O qcow2 -s \$snapshot\_name \$instance-disk**).

On Debian-like distros, you can check the version by running :

```
dpkg -l | grep qemu
```

```
ii  qemu                   0.14.0~rc1+noroms-0ubuntu4~ppalucid1
    dummy transitional package from qemu to qemu
ii  qemu-common            0.14.0~rc1+noroms-0ubuntu4~ppalucid1
    qemu common functionality (bios, documentation)
ii  qemu-kvm               0.14.0~rc1+noroms-0ubuntu4~ppalucid1
    Full virtualization on i386 and amd64 hardware
```

Images can only be created from running instances if Compute is configured to use qcow2 images, which is the default setting. You can explicitly enable the use of qcow2 images by adding the following line to `nova.conf`:

```
use_cow_images=true
```

- **Write data to disk**

Before creating the image, we need to make sure we are not missing any buffered content that wouldn't have been written to the instance's disk. In order to resolve that ; connect to the instance and run `sync` then exit.

- **Create the image**

In order to create the image, we first need obtain the server id :

```
nova list
```

ID	Name	Status	Networks
116	Server 116	ACTIVE	private=20.10.0.14

Based on the output, we run :

```
nova image-create 116 Image-116
```

The command will then perform the image creation (by creating qemu snapshot) and will automatically upload the image to your repository.



### Note

The image that will be created will be flagged as "Private" (For glance : `-is-public=False`). Thus, the image will be available only for the tenant.

- **Check image status**

After a while the image will turn from a "SAVING" state to an "ACTIVE" one.

```
nova image-list
```

will allow you to check the progress :

```
nova image-list
```

ID	Name	Status
20	Image-116	ACTIVE
6	ttylinux-uec-amd64-12.1_2.6.35-22_1-vmlinuz	ACTIVE
7	ttylinux-uec-amd64-12.1_2.6.35-22_1-initrd	ACTIVE
8	ttylinux-uec-amd64-12.1_2.6.35-22_1.img	ACTIVE

- **Create an instance from the image**

You can now create an instance based on this image as you normally do for other images :

```
nova boot --flavor 1 --image 20 New_server
```

- **Troubleshooting**

Mainly, it wouldn't take more than 5 minutes in order to go from a "SAVING" to the "ACTIVE" state. If this takes longer than five minutes, here are several hints:

- The feature doesn't work while you have attached a volume to the instance. Thus, you should detach the volume first, create the image, and re-mount the volume.
- Make sure the version of qemu you are using is not older than the 0.14 version. That would create "unknown option -s" into nova-compute.log.
- Look into nova-api.log and nova-compute.log for extra information.

## Replicating images across multiple data centers

The image service comes with a tool called **glance-replicator** that can be used to populate a new glance server using the images stored in an existing glance server. The images in the replicated glance server preserve the uuids, metadata, and image data from the original. Running the tool will output a set of commands that it supports:

```
$ glance-replicator
Usage: glance-replicator <command> [options] [args]

Commands:

    help <command>    Output help for one of the commands below

    compare           What is missing from the slave glance?
    dump              Dump the contents of a glance instance to local disk.
    livecopy          Load the contents of one glance instance into another.
    load              Load the contents of a local directory into glance.
    size              Determine the size of a glance instance if dumped to disk.

Options:
    --version          show program's version number and exit
    -h, --help          show this help message and exit
    -c CHUNKSIZE, --chunksize=CHUNKSIZE
                        Amount of data to transfer per HTTP write
```

```
-d, --debug          Print debugging information
-D DONTREPLICATE, --dontreplicate=DONTREPLICATE
                    List of fields to not replicate
-m, --metaonly      Only replicate metadata, not images
-l LOGFILE, --logfile=LOGFILE
                    Path of file to log to
-s, --syslog         Log to syslog instead of a file
-t TOKEN, --token=TOKEN
                    Pass in your authentication token if you have one. If
                    you use this option the same token is used for both
                    the master and the slave.
-M MASTERTOKEN, --mastertoken=MASTERTOKEN
                    Pass in your authentication token if you have one.
                    This is the token used for the master.
-S SLAVETOKEN, --slavetoken=SLAVETOKEN
                    Pass in your authentication token if you have one.
                    This is the token used for the slave.
-v, --verbose        Print more verbose output
```

The replicator supports the following commands:

## livecopy: Load the contents of one glance instance into another

```
glance-replicator livecopy fromserver:port toserver:port
```

- *fromserver:port*: the location of the master glance instance
- *toserver:port*: the location of the slave glance instance.

Take a copy of the fromserver, and dump it onto the toserver. Only images visible to the user running the replicator will be copied if glance is configured to use the Identity service (keystone) for authentication. Only images active on *fromserver* are copied across. The copy is done "on-the-wire" so there are no large temporary files on the machine running the replicator to clean up.

## dump: Dump the contents of a glance instance to local disk

```
glance-replicator dump server:port path
```

- *server:port*: the location of the glance instance.
- *path*: a directory on disk to contain the data.

Do the same thing as **livecopy**, but dump the contents of the glance server to a directory on disk. This includes metadata and image data. Depending on the size of the local glance repository, the resulting dump may consume a large amount of local storage. Therefore, we recommend you use the **size** command first to determine the size of the resulting dump.

## load: Load a directory created by the dump command into a glance server

```
glance-replicator load server:port path
```

- *server:port*: the location of the glance instance.
- *path*: a directory on disk containing the data.

Load the contents of a local directory into glance.

The **dump** and **load** are useful when replicating across two glance servers where a direct connection across the two glance hosts is impossible or too slow.

## compare: Compare the contents of two glance servers

```
glance-replicator compare fromserver:port toserver:port
```

- *fromserver:port*: the location of the master glance instance.
- *toserver:port*: the location of the slave glance instance.

The **compare** command will show you the differences between the two servers, which is effectively a dry run of the **livecopy** command.

## size: Determine the size of a glance instance if dumped to disk

```
glance-replicator size
```

- *server:port*: the location of the glance instance.

The **size** command will tell you how much disk is going to be used by image data in either a **dump** or a **livecopy**. Note that this will provide raw number of bytes that would be written to the destination, it has no information about the redundancy costs associated with glance-registry back-ends that use replication for redundancy, such as Swift or Ceph.

## Example using livecopy

Assuming you have a primary glance service running on a node called `primary.example.com` with `glance-api` service running on port 9292 (the default port) and you want to replicate its contents to a secondary glance service running on a node called `secondary.example.com`, also on port 9292, you will first need to get authentication tokens from keystone for the primary and secondary glance server and then you can use the `glance-replicator livecopy` command.

The following example assumes that you have a credentials file for your primary cloud called `primary.openrc` and one for your secondary cloud called `secondary.openrc`.

```
$ source primary.openrc
$ keystone token-get
+-----+-----+
| Property |      Value      |
+-----+-----+
| expires  | 2012-11-16T03:13:08Z
| id       | 8a5d3afb5095430891f33f69a2791463
| tenant_id | dba21b41af584daeac5782ca15a77a25
| user_id   | add2ece6b1f94866994d3a3e3beb3d47
```

```
+-----+-----+
$ PRIMARY_AUTH_TOKEN=8e97fa8bcf4443cfbd3beb9079c7142f
$ source secondary.openrc
$ keystone token-get
+-----+-----+
| Property |          Value          |
+-----+-----+
| expires  | 2012-11-16T03:13:08Z |
| id       | 29f777ac2c9b41a6b4ee9c3e6b85f98a |
| tenant_id | fbde89d638d947a19545b0f387ffea4d |
| user_id  | 4a7a48e7d62e4b428c78d02c1968ca7b |
+-----+-----+
$ SECONDARY_AUTH_TOKEN=29f777ac2c9b41a6b4ee9c3e6b85f98a
$ glance-replicator livecopy primary.example.com:9292 secondary.example.
com:9292 -M ${PRIMARY_AUTH_TOKEN} -S ${SECONDARY_AUTH_TOKEN}
```

# 8. Instance Management

## Table of Contents

Interfaces to managing instances .....	145
Instance building blocks .....	147
Creating instances .....	150
Controlling where instances run .....	152
Instance specific data .....	153
Configuring instances at boot time .....	155
Config drive .....	158
Managing instance networking .....	162
Manage Volumes .....	167
Accessing running instances .....	168
Stop and Start an Instance .....	168
Change Server Configuration .....	168
Instance evacuation .....	171
Terminate an Instance .....	172

Instances are the running virtual machines within an OpenStack cloud. The [Images and Instances](#) section of the [Introduction to OpenStack Compute](#) Chapter provides a high level overview of instances and their life cycle

This chapter deals with the details of how to manage that life cycle

## Interfaces to managing instances

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

### Nova CLI

The nova command provided by the OpenStack python-novaclient package is the basic command line utility for users interacting with OpenStack. This is available as a native package for most modern Linux distributions or the latest version can be installed directly using pip python package installer:

```
sudo pip install -e git+https://github.com/openstack/python-novaclient.  
git#egg=python-novaclient
```

Full details for nova and other CLI tools are provided in the [OpenStack CLI Guide](#). What follows is the minimal introduction required to follow the CLI example in this chapter. In the case of a conflict the [OpenStack CLI Guide](#) should be considered authoritative (and a bug filed against this section).

In order to function the nova CLI needs to know four things:

- Authentication URL. This can be passed as the `-os_auth_url` flag or using the `OS_AUTH_URL` environment variable.
- Tenant(sometimes referred to as project) name. This can be passed as the `-os_tenant_name` flag or using the `OS_TENANT_NAME` environment variable.
- User name. This can be passed as the `-os_username` flag or using the `OS_USERNAME` environment variable.
- Password. This can be passed as the `-os_password` flag or using the `OS_PASSWORD` environment variable.

For example if you have your [Keystone](#) identity management service running on the default port (5000) on host `keystone.example.com` and want to use the `nova` cli as the user "demouser" with the password "demopassword" in the "demoproject" tenant you can export the following values in your shell environment or pass the equivalent command line args (presuming these identities already exist):

```
export OS_AUTH_URL="http://keystone.example.com:5000/v2.0/"
export OS_USERNAME=demouser
export OS_PASSWORD=demopassword
export OS_TENANT_NAME=demoproject
```

If you are using the [Horizon](#) web dashboard, users can easily download credential files like this with the correct values for your particular implementation.

## Horizon web dashboard

Horizon is the highly customizable and extensible OpenStack web dashboard. The [Horizon Project](#) home page has detailed information on deploying horizon.

## Compute API

OpenStack provides a RESTful API for all functionality. Complete API documentation is available at <http://docs.openstack.org/api>. The [OpenStack Compute API](#) documentation refers to instances as "servers".

The [nova cli](#) can be made to show the API calls it is making by passing it the `--debug` flag

```
#nova --debug list
connect: (10.0.0.15, 5000)
send: 'POST /v2.0/tokens HTTP/1.1\r\nHost: 10.0.0.15:5000\r\nContent-Length:
116\r\nContent-Type: application/json\r\nAccept-Encoding: gzip, deflate\r\n
Accept: application/json\r\nUser-Agent: python-novaclient\r\n\r\n{"auth":
{"tenantName": "demoproject", "passwordCredentials": {"username": "demouser",
"password": "demopassword"}}}'
reply: 'HTTP/1.1 200 OK\r\n'
header: Content-Type: application/json
header: Vary: X-Auth-Token
header: Date: Thu, 13 Sep 2012 20:27:36 GMT
header: Transfer-Encoding: chunked
connect: (128.52.128.15, 8774)
send: u'GET /v2/fa9dccdeadbeef23ae230969587a14bf/servers/detail HTTP/1.1\
\r\nHost: 10.0.0.15:8774\r\nX-Auth-Project-Id: demoproject\r\nX-Auth-Token:
```

```
deadbeef9998823afecc3d552525c34c\r\naccept-encoding: gzip, deflate\r\naccept:  
application/json\r\nuser-agent: python-novaclient\r\n\r\n'  
reply: 'HTTP/1.1 200 OK\r\n'  
header: X-Compute-Request-Id: req-bf313e7d-771a-4c0b-ad08-c5da8161b30f  
header: Content-Type: application/json  
header: Content-Length: 15  
header: Date: Thu, 13 Sep 2012 20:27:36 GMT  
+-----+-----+-----+  
| ID | Name | Status | Networks |  
+-----+-----+-----+  
+-----+-----+-----+
```

## EC2 Compatibility API

In addition to the native compute API OpenStack provides an EC2 compatible API. This allows legacy workflows built for EC2 to work with OpenStack.

[Configuring the EC2 API](#) lists configuration options for customizing this compatibility API on your OpenStack cloud.

## Third Party Tools

There are numerous third party tools and language specific SDKs for interacting with OpenStack clouds both through native and compatibility APIs. These are not OpenStack projects so we can only provide links to some of the more popular projects and a brief description. For detailed installation and usage info please see the individual project pages

- [euca2ools](#) is a popular open source CLI for interacting with the EC2 API. This is convenient for multi cloud environments where EC2 is the common API, or for transitioning from EC2 API based clouds to OpenStack.
- [hybridfox](#) is a Firefox browser add-on that provides a graphical interface to many popular public and private cloud technologies.
- [boto](#) is a Python library for interacting with Amazon Web Services. It can be used to access OpenStack through the EC2 compatibility API
- [fog](#) is the Ruby cloud services library and provides methods for interacting with a large number of cloud and virtualization platforms.
- [heat](#) is a high level orchestration system that provides a programmable interface to orchestrate multiple cloud applications implementing well known standards such as CloudFormation and TOSCA. Unlike other projects mentioned in this section [heat](#) requires changes to your OpenStack deployment and is working toward official inclusion as an OpenStack project. At this point [heat](#) is a development project not a production resource, but it does show what the not too distant future of instance management may be like.

## Instance building blocks

There are two fundamental requirements for a computing system, software and hardware. Virtualization and cloud frameworks tend to blur these lines and some of your "hardware"

may actually be "software" but conceptually you still need an operating system and something to run it on.

## Images

In OpenStack the base operating system is usually copied from an "[image](#)" stored in the Glance image service. This is the most common case and results in an ephemeral instance which starts from a known templated state and loses all accumulated state on shutdown. It is also possible in special cases to put an operating system on a persistent "volume" in the Nova-Volume or Cinder volume system. This gives a more traditional persistent system that accumulates state which is preserved across restarts. To get a list of available images on your system run:

```
$nova image-list
```

ID	Status	Server	Name	
ae1d242-730f-431f-88c1-87630c0f07ba	ACTIVE		Ubuntu 12.04 cloudimg amd64	
0b27baa1-0ca6-49a7-b3f4-48388e440245	ACTIVE		Ubuntu 12.10 cloudimg amd64	
df8d56fc-9cea-4dfd-a8d3-28764de3cb08	ACTIVE		jenkins	

The displayed image attributes are

- ID: the automatically generated UUID of the image
- Name: a free form human readable name given to the image
- Status: shows the status of the image ACTIVE images 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 it is blank

## Flavors

Virtual hardware templates are called "flavors" in OpenStack. The default install provides a range of five flavors. These are configurable by admin users (this too is configurable and may be delegated by redefining the access controls for "compute\_extension:flavormanage" in /etc/nova/policy.json on the compute-api server). To get a list of available flavors on your system run:

```
$ nova flavor-list
```

|--|--|--|--|--|--|--|

ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor
Is_Public	extra_specs						
1	m1.tiny	512	0	0		1	1.0
True	{}						
2	m1.small	2048	10	20		1	1.0
True	{}						
3	m1.medium	4096	10	40		2	1.0
True	{}						
4	m1.large	8192	10	80		4	1.0
True	{}						
5	m1.xlarge	16384	10	160		8	1.0
True	{}						

The **nova flavor-create** command allows authorized users to create new flavors. Additional flavor manipulation commands can be shown with the command **nova help |grep flavor**

Flavors define a number of elements

- ID: a unique numeric id
- Name: a descriptive name. *xx.size\_name* is conventional not required, though some third party tools may rely on it.
- Memory\_MB: virtual machine memory in megabytes
- Disk: virtual root disk size in gigabytes. This is an ephemeral disk the base [image](#) is copied into. When booting from a persistent volume it is not used. The "0" size is a special case which uses the native base image size as the size of the ephemeral root volume.
- Ephemeral: specifies the size of a secondary ephemeral data disk. This is an empty, unformatted disk and exists only for the life of the instance.
- Swap: optional swap space allocation for the instance
- VCPUs: number of virtual CPUs presented to the instance
- RXTX\_Factor: optional property allows created servers to have a different bandwidth cap than that defined in the network they are attached to. This factor is multiplied by the rxtx\_base property of the network. Default value is 1.0 (that is, the same as attached network).
- Is\_Public: Boolean value, whether flavor is available to all users or private to the tenant it was created in. Defaults to True.
- extra\_specs: additional optional restrictions on which compute nodes the flavor can run on. This is implemented as key/value pairs that must match against the corresponding key/value pairs on compute nodes. Can be used to implement things like special resources (e.g., flavors that can only run on compute nodes with GPU hardware).

# Creating instances

## Create Your Server with the nova Client

### Procedure 8.1. To create and boot your server with the nova client:

- Issue the following command. In the command, specify the server name, flavor ID, and image ID:

```
$ nova boot myUbuntuServer --image "3afe97b2-26dc-49c5-a2cc-a2fc8d80c001" --flavor 6
```

The command returns a list of server properties. The status field indicates whether the server is being built or is active. A status of `BUILD` indicates that your server is being built.

Property	Value
OS-DCF:diskConfig	AUTO
accessIPv4	
accessIPv6	
adminPass	ZbaYPZf6r2an
config_drive	
created	2012-07-27T19:59:31Z
flavor	8GB Standard Instance
hostId	
id	d8093de0-850f-4513-b202-7979de6c0d55
image	Ubuntu 12.04
metadata	{}
name	myUbuntuServer
progress	0
status	BUILD
tenant_id	345789
updated	2012-07-27T19:59:31Z
user_id	170454

- Copy the server ID value from the `id` field in the output. You use this ID to get details for your server to determine if it built successfully.

Copy the administrative password value from the `adminPass` field. You use this value to log into your server.

## Launch from a Volume

The Compute service has support for booting an instance from a volume.

### Manually Creating a Bootable Volume

To manually create a bootable volume, mount the volume to an existing instance, and then build a volume-backed image. Here is an example based on [exercises/boot\\_from\\_volume.sh](#). This example assumes that you have a running instance with a 1GB volume mounted at `/dev/vdc`. These commands will make the mounted volume bootable using a CirrOS image. As root:

```
# mkfs.ext3 -b 1024 /dev/vdc 1048576
# mkdir /tmp/stage
# mount /dev/vdc /tmp/stage

# cd /tmp
# wget https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-
rootfs.img.gz
# gunzip cirros-0.3.0-x86_64-rootfs.img.gz
# mkdir /tmp/cirros
# mount /tmp/cirros-0.3.0-x86_64-rootfs.img /tmp/cirros

# cp -pr /tmp/cirros/* /tmp/stage
# umount /tmp/cirros
# sync
# umount /tmp/stage
```

Detach the volume once you are done.

## Creating a Bootable Volume from an Image

Cinder has the ability to create a bootable volume from an image stored in Glance.

```
# cinder create --image-id <image_id> --display-name my-bootable-vol <size>
```

This feature is also mirrored in Nova:

```
# nova volume-create --image-id <image_id> --display-name my-bootable-vol
<size>
```



### Note

As of Grizzly, the following block storage drivers are compatible: iSCSI-based, LVM, and Ceph.

Make sure you configure Cinder with the relevant Glance options:

**Table 8.1. List of configuration flags for NFS**

Flag Name	Type	Default	Description
glance_host	Optional	\$my_ip	(StrOpt) default glance hostname or ip
glance_port	Optional	9292	(IntOpt) default glance port
glance_api_servers	Optional	\$glance_host:\$glance_port	(ListOpt) A list of the glance api servers available to cinder: ([hostname ip]:port) (list value)
glance_api_version	Optional	1	(IntOpt) default version of the glance api to use
glance_num_retries	Optional	0	(IntOpt) Number retries when downloading an image from glance
glance_api_insecure	Optional	false	(BoolOpt) Allow to perform insecure SSL (https) requests to glance

## Booting an instance from the volume

To boot a new instance from the volume, use the **nova boot** command with the **--block\_device\_mapping** flag. The output for **nova help boot** shows the following documentation about this flag:

```
--block_device_mapping <dev_name=mapping>
    Block device mapping in the format <dev_name>=
    <id>:<type>:<size(GB)>:<delete_on_terminate>.
```

The command arguments are:

dev_name	A device name where the volume will be attached in the system at <code>/dev/dev_name</code> . This value is typically vda.
id	The ID of the volume to boot from, as shown in the output of <b>nova volume-list</b> .
type	This is either <code>snap</code> , which means that the volume was created from a snapshot, or anything other than <code>snap</code> (a blank string is valid). In the example above, the volume was not created from a snapshot, so we will leave this field blank in our example below.
size (GB)	The size of the volume, in GB. It is safe to leave this blank and have the Compute service infer the size.
delete_on_terminate	A boolean to indicate whether the volume should be deleted when the instance is terminated. True can be specified as <code>True</code> or <code>1</code> . False can be specified as <code>False</code> or <code>0</code> .



### Note

Because of bug [#1163566](#), you must specify an image when booting from a volume in Horizon, even though this image will not be used.

The following example will attempt boot from volume on the command line with ID=13, it will not delete on terminate. Replace the `--key_name` with a valid keypair name:

```
$ nova boot --flavor 2 --key_name mykey --block_device_mapping vda=13:::0
boot-from-vol-test
```

## Controlling where instances run

The [scheduler filters](#) section 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

Additionally admin users can specify an exact compute node to run on by specifying `--availability-zone <availability-zone>:<compute-host>` on the command line, for example to force an instance to launch on the `nova-1` compute node in the default `nova` availability zone:

```
#nova boot --image ae1d242-730f-431f-88c1-87630c0f07ba --flavor 1 --  
availability-zone nova:nova-1 testhost
```

## Instance specific data

For each instance, you can specify certain data including authorized\_keys key injection, user-data, metadata service, and file injection.

## Associating ssh keys with instances

### Creating New Keys

The command:

```
$ nova keypair-add mykey > mykey.pem
```

will create a key named `mykey` which you can associate with instances. Save the file `mykey.pem` to a secure location as it will allow root access to instances the `mykey` is associated with.

### Uploading Existing Keys

The command:

```
$ nova keypair-add --pub-key mykey.pub mykey
```

will upload the existing public key `mykey.pub` and associate it with the name `mykey`. You will need to have the matching private key to access instances associated with this key.

### Adding Keys to Your Instance

To associate a key with an instance on boot add `--key_name mykey` to your command line for example:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --key_name mykey
```

### Insert metadata during launch

When booting a server, you can also add metadata, so that you can more easily identify it amongst your ever-growing elastic cloud. Use the `--meta` option with a key=value pair, where you can make up the string for both the key and the value. For example, you could add a description and also the creator of the server.

```
$ nova boot --image=natty-image --flavor=2 smallimage2 --meta description=  
'Small test image' --meta creator=joe cool
```

When viewing the server information, you can see the metadata included on the metadata line:

```
$ nova show smallimage2
+-----+
| Property | Value |
+-----+
| OS-DCF:diskConfig | MANUAL |
| OS-EXT-STS:power_state | 1 |
| OS-EXT-STS:task_state | None |
| OS-EXT-STS:vm_state | active |
| accessIPv4 | |
| accessIPv6 | |
| config_drive | |
| created | 2012-05-16T20:48:23Z |
| flavor | m1.small |
| hostId | de0c201e62be88c61aeb52f51d91e147acf6cf2012bb57892e528487 |
| id | 8ec95524-7f43-4cce-a754-d3e5075bf915 |
| image | natty-image |
| key_name | |
| metadata | {u'description': u'Small test image', u'creator': u'joe cool'} |
| name | smallimage2 |
| private network | 172.16.101.11 |
| progress | 0 |
| public network | 10.4.113.11 |
| status | ACTIVE |
| tenant_id | e830c2fbb7aa4586adf16d61c9b7e482 |
| updated | 2012-05-16T20:48:35Z |
| user_id | de3f4e99637743c7b6d27faca4b800a9 |
+-----+
```

## Providing User Data to Instances

User Data is a special key in the metadata service which holds a file that cloud aware applications within the guest instance can access. For example the [cloudinit](#) system is an open source package from Ubuntu that handles early initialization of a cloud instance that makes use of this user data.

This user-data can be put in a file on your local system and then passed in at instance creation with the flag `--user-data <user-data-file>` for example:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --user-data mydata.file
```

## Injecting Files into Instances

Arbitrary local files can also be placed into the instance file system at creation time using the `--file <dst-path=src-path>` option. You may store up to 5 files. For example if you have a special authorized\_keys file named `special_authorized_keysfile` that you want to put on the instance rather than using the regular [ssh key injection](#) for some reason you can use the following command:

```
$nova boot --image ubuntu-cloudimage --flavor 1 --file /root/.ssh/
authorized_keys=special_authorized_keysfile
```

# Configuring instances at boot time

## Introduction

Users often want to do some configuration to their instances after booting. For example, you may want to install some packages, start services, or manage the instance using a Puppet or Chef server. When launching instances in an OpenStack cloud, there are two technologies that work together to support automated configuration of instances at boot time: user data and cloud-init.

## User data

User data is the mechanism by which a user can pass information contained in a local file to an instance at launch time. The typical use case is to pass something like a shell script or a configuration file as user data.

User data is sent using the `--user-data /path/to/filename` option when calling `nova boot`. The following example creates a text file and then send the contents of that file as user data to the instance.

```
$ echo "This is some text" > myfile.txt
$ nova boot --user-data ./myfile.txt --image myimage myinstance
```

The instance can retrieve user data by querying the metadata service at using either the OpenStack metadata API or the EC2 compatibility API:

```
$ curl http://169.254.169.254/2009-04-04/user-data
This is some text
$ curl http://169.254.169.254/openstack/2012-08-10/user_data
This is some text
```

Note that the Compute service treats user data as a blob. While the example above used a text file, user data can be in any format.

## Cloud-init

To do something useful with the user data, the virtual machine image must be configured to run a service on boot that retrieves the user data from the metadata service and take some action based on the contents of the data. The cloud-init package was designed to do exactly this. In particular, cloud-init is compatible with the Compute metadata service as well as the Compute config drive.

Note that cloud-init is not an OpenStack technology. Rather, it is a package that is designed to support multiple cloud providers, so that the same virtual machine image can be used in different clouds without modification. Cloud-init is an open source project, and the source code is available on [Launchpad](#). It is maintained by Canonical, the company which runs the Ubuntu project. All Ubuntu cloud images come pre-installed with cloud-init. However, cloud-init is not designed to be Ubuntu-specific, and has been successfully ported to Fedora.

We recommend installing cloud-init on images that you create to simplify the task of configuring your instances on boot. Even if you do not wish to use user data to configure instance behavior at boot time, cloud-init provides useful functionality such as copying the

public key to an account (the `ubuntu` account by default on Ubuntu instances, the `ec2-user` by default in Fedora instances).

If you do not have cloud-init installed, you will need to manually configure your image to retrieve the public key from the metadata service on boot and copy it to the appropriate account.

## Cloud-init supported formats and documentation

We recommend taking a look at the cloud-init [doc/userdata.txt](#) file the [examples](#) directory as well as the [Ubuntu community documentation](#) for details on how to use cloud-init. We provide some basic examples here.

Cloud-init supports several different input formats for user data. We briefly discuss two commonly used formats:

- Shell scripts (starts with `# !`)
- Cloud config files (starts with `#cloud-config`)

## Running a shell script on boot

Assuming you have cloud-init installed, the simplest way to configure an instance on boot is to pass a shell script as user data. The shell file must begin with `# !` in order for cloud-init to recognize it as a shell script. Here's an example of a script that creates an account called `clouduser`.

```
#!/bin/bash
adduser --disabled-password --gecos "" clouduser
```

Sending a shell script as user data has a similar effect to writing an `/etc/rc.local` script: it will be executed very late in the boot sequence as root.

## Cloud-config format

Cloud-init supports a YAML-based config format that allows the user to configure a large number of options on a system. User data that begins with `#cloud-config` will be interpreted by cloud-init as cloud-config format.

## Example: Setting hostname

This cloud-init user data example sets the hostname and the FQDN, as well as updating `/etc/hosts` on the instance:

```
#cloud-config
hostname: mynode
fqdn: mynode.example.com
manage_etc_hosts: true
```

## Example: Configuring instances with Puppet

This cloud-init user data example, based on [doc/examples/cloud-config-puppet.txt](#), would configure the instance to contact a Puppet server at `puppetmaster.example.org` and verify its identity using a certificate.

```
#cloud-config
puppet:
  conf:
    agent:
      server: "puppetmaster.example.org"
    ca_cert: |
      -----BEGIN CERTIFICATE-----
MIICCTCCAXKgAwIBAgIBATANBgkqhkiG9w0BAQUFADANMQswCQYDVQQDDAJjYTAe
Fw0xMDAyMTUxNzI5MjFaFw0xNTAyMTQxNzI5MjFaMA0xCzAJBgNVBAMMAnhMIGf
MA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCu7Q40sm47/E1Pf+r8AYb/V/FWGPgc
b014OmNoX7dgCxTDvps/h8Vw555PdAFsW5+QhsGr31IJN13kSYprFQcYf7A8tNWu
1MASW2CfaEiOEi9F1R3R4Qlz4ix+iNoHiUDTjazw/tZwEdxaQXQVLwgTGRwVa+aA
qbutJKi93MILLwIDAQABo3kwdaA4BglghkgBhvhaCAQ0EKxYpUHVwcGV0IFJ1Ynkv
T3B1blNTTCBHZW51cmF0ZWQgQ2VydG1maWNhdGUwDwYDVR0TAQH/BAUwAwEB/zAd
BgNVHQ4EFgQUu4+jHB+GYE5Vxo+o110AhevspjAwCwYDVR0PBAQDAgEGMA0GCSqG
S1b3DQEBBQUAA4GBAH/rxlUIjwNb3n7TXJcDJ6MMHU1wjr03BDJXKb34Ulndkpaf
+GA1zPXWa7b0908M9I8RnPfvtknteLbvgTK+h+zX1Xcty+S2EQWk29i2AdoqOTxb
hppiGMp0tT5Havu4aceCXiy2crVcudj3NFciy8X66SoECemW9UYDCb9T5D0d
-----END CERTIFICATE-----
```

## Example: Configuring instances with Chef

This cloud-init user data example, based on [doc/examples/cloud-config/chef.txt](#), and intended for use in an Ubuntu image, would add the Chef apt repository, install Chef, connect to a Chef server at <https://chefserver.example.com:4000> and install Apache.

```
#cloud-config
apt_sources:
- source: "deb http://apt.opscode.com/ $RELEASE-0.10 main"
  key: |
    -----BEGIN PGP PUBLIC KEY BLOCK-----
Version: GnuPG v1.4.9 (GNU/Linux)

mQGiBEppC7QRBADfsOkZU6KZK+YmKw4wev5mjKJEkVGlus+NxW8wItX5sGa6kdUu
twAyj7Yr92rF+ICFEP3gGU6+1Go0Nve7KxkN/1W7/m3G4zuk+ccIKmjp8KS3qn99
dxy64vcji9jI11Va+XXOGIp0G8Geaj7mbkixL/bMeGfdM1v8Gf2XPpp9vwCgn/GC
JKacfwn7MpLKH0YS1b/JsEAJqao3ViNfav83jjKEkD8cf59Y8xKia5OpZqTK5W
ShVnNWS3U5IVQk10ZDH97Qn/YrK387H4CyhLE9mxPXs/u118ioiaars/q2MEKU2I
XKfV21eMLO9Lyd6Ny/Kqj8o5WQK2J6+NAhSwvthZcIEphcFignIuobP+B5wNFQpe
DbkFA/0WvN2OwFeWRcmm3Hz7nHTpcnSF+4QX6yHRF/5BgxkG61qBIACQbzPn6Hm
sMtm/SVf11izmDqSsQptCrOZILfLX/mE+Y01+CwWSHh1+YsFts1WOuh1EhQD26aO
Z84HuHV5HFRWjDLw9Lrl1tBVQcXbpfSrRP5bdr7Wh8vhqJTPjrQnT3BzY29kZSBQ
YWNrYWdlcyA8cGFja2FnZXNAb3BzY29kZS5jb20+iGAEEexECACAFAkppC7QCGwMG
CwkIBwMCBBUCAMEFgIDAQIeAQIXgAAKCRApQKupg++Ca j8sAKCOXmdG36gWji/K
+o+xtBfvdMnFYQCfTCEWxRy2BnzLoBBFCjDSK6sJqCu5Ag0ESmkLtBAIAIO2Sw1R
1U5i6gT0p42RHWW7/pmW78CwUqJnYqnxROrt3h9F9xrsGkH0Fh1FRtsnnncgzIhv
DLQnRHnkXm0ws0jv0PF74tt0UT6BLAUsFi2SPP1zYNJ9H9fhk/pjijtAcQwdgxu
wwNJ5xCEscBZCjhSRxm0d30bK1o49Cow8Z1bHtnXVP41c9QWoZx/LaGZsKQZnaMx
EzDk8dyyctR2f03vRSVyTFGgdpUcpbr9eTFVgikCa6ODEBv+0BnCH6yGTXwBid9g
w0ole/2DviKUWCC+A1AUoubLmOIGFBuI4UR+rux9affbHcLIOTiKQXv791W3P7W8
AAfnisQKfPWxrrcAAwUH/2XBqD4Uxhbs25HDUU1m/m6Gn1j6EsStg8n0nMggLhuN
QmPfoNByMPUqvA7sULyfr6xCYzbzRNxABHSpf85FzGQ29RF4xsA4vOOU8RD1YQ9X
Q8NqqR6pydprRFqWe47hsAN7BoYuhWqTtOLSbmnnAnzTR5pUroqcquWYiiEavZixJ
3ZRAq/HMGioJEtMFrvsZjGXuzef7f0ytfr1zYeLVWnL9Bd32CueB1I7dhYwkFe+v
Ep5jWOCj02C1wHcwt+uIRDJV6TdtbIiBYAdOMPk15+VBdwEBxwMuYXr76+A7VeDL
zIhi7tKFo6WiwjKZq0dzctsJJjtIfra4K4vbiD90jg1iISQQYEQIAcQUCSmkLtA1b
DAAKCRApQKupg++CauISAJ9CxYPOKhOxalBnVTLLeNUkAHGg2gACeIsbobtaD4ZHg
0GLl8Ekfa8uhluM=
```

```
=zKAm
-----END PGP PUBLIC KEY BLOCK-----

chef:
  install_type: "packages"
  server_url: "https://chefserver.example.com:4000"
  node_name: "your-node-name"
  environment: "production"
  validation_name: "yourorg-validator"
  validation_key: |
    -----BEGIN RSA PRIVATE KEY-----
    YOUR-ORGS-VALIDATION-KEY-HERE
    -----END RSA PRIVATE KEY-----
  run_list:
    - "recipe[apache2]"
    - "role[db]"
  initial_attributes:
    apache:
      prefork:
        maxclients: 100
        keepalive: "off"
```

## Config drive

### Introduction

OpenStack can be configured to write metadata to a special configuration drive that will be attached to the instance when it boots. The instance can retrieve any information that would normally be available through the [metadata service](#) by mounting this disk and reading files from it.

One use case for the config drive is to pass networking configuration (e.g., IP address, netmask, gateway) when DHCP is not being used to assign IP addresses to instances. The instance's IP configuration can be transmitted using the config drive, which can be mounted and accessed before the instance's network settings have been configured.

The config drive can be used by any guest operating system that is capable of mounting an ISO9660 or VFAT file system. This functionality should be available on all modern operating systems.

In addition, an image that has been built with a recent version of the cloud-init package will be able to automatically access metadata passed via config drive. The current version of cloud-init as of this writing (0.7.1) has been confirmed to work with Ubuntu, as well as Fedora-based images such as RHEL.

If an image does not have the cloud-init package installed, the image must be customized to run a script that mounts the config drive on boot, reads the data from the drive, and takes appropriate action such as adding the public key to an account. See below for details on how data is organized on the config drive.



#### Note

To use config drive, the `genisoimage` program must be installed on each compute host. In the Ubuntu packages, it is not installed by default (see bug

#1165174). Make sure you install this program on each compute host before attempting to use config drive, or an instance will not boot properly.

## Enabling the config drive

To enable the config drive, pass the `--config-drive=true` parameter when calling `nova boot`. Here is a complex example that enables the config drive as well as passing user data, two files, and two key/value metadata pairs, all of which will be accessible from the config drive as described below.

```
$ nova boot --config-drive=true --image my-image-name --key-name mykey --flavor 1 --user-data ./my-user-data.txt myinstance --file /etc/network/interfaces=/home/myuser/instance-interfaces --file known_hosts=/home/myuser/.ssh/known_hosts --meta role=webservers --meta essential=false
```

You can also configure the Compute service to always create a config drive by setting the following option in `/etc/nova/nova.conf`:

```
force_config_drive=true
```



### Note

As of this writing, there is no mechanism for an administrator to disable use of the config drive if a user passes the `--config-drive=true` flag to the `nova boot` command.

## Accessing the config drive from inside an instance

The config drive will have a volume label of `config-2`. If your guest OS supports accessing disk by label, you should be able to mount the config drive as the `/dev/disk/by-label/config-2` device. For example:

```
# mkdir -p /mnt/config
# mount /dev/disk/by-label/config-2 /mnt/config
```



### Note

The cirros 0.3.0 test image does not have support for the config drive. Support will be added in version 0.3.1.

If your guest operating system does not use udev, then the `/dev/disk/by-label` directory will not be present. The `blkid` command can be used to identify the block device that corresponds to the config drive. For example, when booting the cirros image with the `m1.tiny` flavor, the device will be `/dev/vdb`:

```
# blkid -t LABEL="config-2" -odevice
/dev/vdb
```

Once identified, the device can then be mounted:

```
# mkdir -p /mnt/config
# mount /dev/vdb /mnt/config
```

## Contents of the config drive

The files that will be present in the config drive will vary depending on the arguments that were passed to **nova boot**. Based on the example above, the contents of the config drive would be:

```
ec2/2009-04-04/meta-data.json
ec2/2009-04-04/user-data
ec2/latest/meta-data.json
ec2/latest/user-data
openstack/2012-08-10/meta_data.json
openstack/2012-08-10/user_data
openstack/content
openstack/content/0000
openstack/content/0001
openstack/latest/meta_data.json
openstack/latest/user_data
```

## Guidelines for accessing config drive data

Do not rely on the presence of the EC2 metadata present in the config drive (i.e., files under the `ec2` directory), as this content may be removed in a future release.

When creating images that access config drive data, if there are multiple directories under the `openstack` directory, always select the highest API version by date that your consumer supports. For example, if your guest image can support versions 2012-03-05, 2012-08-05, 2013-04-13. It is best to try 2013-04-13 first and fall back to an earlier version if it 2013-04-13 isn't present.

## Format of OpenStack metadata

Here is an example of the contents of `openstack/2012-08-10/meta_data.json`, `openstack/latest/meta_data.json` (these two files are identical), formatted to improve readability:

```
{
    "availability_zone": "nova",
    "files": [
        {
            "content_path": "/content/0000",
            "path": "/etc/network/interfaces"
        },
        {
            "content_path": "/content/0001",
            "path": "known_hosts"
        }
    ],
    "hostname": "test.novalocal",
    "launch_index": 0,
    "name": "test",
    "meta": {
        "role": "webservers",
        "essential": "false"
    }
},
```

```
    "public_keys": {
        "mykey": "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAAAgQDBqUfVvCSez0/
Wfpd8dLLgZXV9GtXQ7hnMN+Z0OWQUyebVEHey1CXuin0uY1cAJMhUq8j98SiW
+cu0sU4J3x5l2+xi1bodDm1BtFWVeLIOQINpfVln8fKjHB
+ynPpe1F6tMDvrfGULJs44t30BrujMXBe8Rq44cCk6wqyjATA3rQ== Generated by Nova\n"
    },
    "uuid": "83679162-1378-4288-a2d4-70e13ec132aa"
}
```

Note the effect of the `--file /etc/network/interfaces=/home/myuser/instance-interfaces` argument passed to the original **nova boot** command. The contents of this file are contained in the file `openstack/content/0000` file on the config drive, and the path is specified as `/etc/network/interfaces` in the `meta_data.json` file.

## Format of EC2 metadata

Here is an example of the contents of `ec2/2009-04-04/meta-data.json`, `latest/meta-data.json` (these two files are identical) formatted to improve readability:

```
{
    "ami-id": "ami-00000001",
    "ami-launch-index": 0,
    "ami-manifest-path": "FIXME",
    "block-device-mapping": {
        "ami": "sdal",
        "ephemeral0": "sda2",
        "root": "/dev/sdal",
        "swap": "sda3"
    },
    "hostname": "test.novalocal",
    "instance-action": "none",
    "instance-id": "i-00000001",
    "instance-type": "m1.tiny",
    "kernel-id": "aki-00000002",
    "local-hostname": "test.novalocal",
    "local-ipv4": null,
    "placement": {
        "availability-zone": "nova"
    },
    "public-hostname": "test.novalocal",
    "public-ipv4": "",
    "public-keys": {
        "0": {
            "openssh-key": "ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAAAgQDBqUfVvCSez0/
Wfpd8dLLgZXV9GtXQ7hnMN+Z0OWQUyebVEHey1CXuin0uY1cAJMhUq8j98SiW
+cu0sU4J3x5l2+xi1bodDm1BtFWVeLIOQINpfVln8fKjHB
+ynPpe1F6tMDvrfGULJs44t30BrujMXBe8Rq44cCk6wqyjATA3rQ== Generated by Nova\n"
        }
    },
    "ramdisk-id": "ari-00000003",
    "reservation-id": "r-7lfps8wj",
    "security-groups": [
        "default"
    ]
}
```

## User data

The files `openstack/2012-08-10/user_data`, `openstack/latest/user_data`, `ec2/2009-04-04/user-data`, and `ec2/latest/user-data`, will only be present if the `--user-data` flag was passed to `nova boot` and will contain the contents of the user data file passed as the argument.

## Format of the config drive

The default format of the config drive as an ISO 9660 filesystem. To explicitly specify the ISO 9660 format, add the following line to `/etc/nova/nova.conf`:

```
config_drive_format=iso9660
```

For legacy reasons, the config drive can be configured to use VFAT format instead of ISO 9660. It is unlikely that you would require VFAT format, since ISO 9660 is widely supported across operating systems. However, if you wish to use the VFAT format, add the following line to `/etc/nova/nova.conf` instead:

```
config_drive_format=vfat
```

If VFAT is chosen, the config drive will be 64MB in size.

## Managing instance networking

### Manage Floating IP Addresses

A floating IP address is an IP address (typically public) that can be dynamically assigned to an instance. Pools of floating IP addresses are created outside of `python-novaclient` with the `nova-manage floating *` commands. Refer to "Configuring Public (Floating) IP Addresses" in the *OpenStack Compute Administration Manual* for more information.

Before you begin, use `nova floating-ip-pool-list` to determine what floating IP pools are available.

```
$ nova floating-ip-pool-list
+----+
| name |
+----+
| nova |
+----+
```

In this example, the only available pool is `nova`.

### Reserve and associate floating IP addresses

You can reserve floating IP addresses with the `nova floating-ip-create` command. This command reserves the addresses for the tenant, but does not immediately associate that address with an instance.

```
$ nova floating-ip-create nova
+-----+-----+-----+-----+
|     Ip      | Instance Id | Fixed Ip | Pool |
+-----+-----+-----+-----+
| 50.56.12.232 |      None    |      None  | nova  |
+-----+-----+-----+-----+
```

The floating IP address has been reserved, and can now be associated with an instance with the **nova add-floating-ip** command. For this example, we'll associate this IP address with an image called `smallimage`.

```
$ nova add-floating-ip smallimage 50.56.12.232
```

After the command is complete, you can confirm that the IP address has been associated with the **nova floating-ip-list** and **nova-list** commands.

```
$ nova floating-ip-list
+-----+-----+-----+-----+
|     Ip      |           Instance Id          | Fixed Ip | Pool |
+-----+-----+-----+-----+
| 50.56.12.232 | 542235df-8ba4-4d08-90c9-b79f5a77c04f | 10.4.113.9 | nova |
+-----+-----+-----+-----+

$ nova list
+-----+-----+
+-----+-----+
|       ID      |       Name    | Status |
| Networks           |           |
+-----+-----+
+-----+-----+
| 4bb825ea-ea43-4771-a574-ca86ab429dc | tinyimage2 | ACTIVE | public=10.4.113.6; private=172.16.101.6 |
| 542235df-8ba4-4d08-90c9-b79f5a77c04f | smallimage | ACTIVE | public=10.4.113.9, 50.56.12.232; private=172.16.101.9 |
+-----+-----+
+-----+-----+
```

The first table shows that the 50.56.12.232 is now associated with the `smallimage` instance ID, and the second table shows the IP address included under `smallimage`'s public IP addresses.

## Remove and de-allocate a floating IP address

To remove a floating IP address from an instance, use the **nova remove-floating-ip** command.

```
$ nova remove-floating-ip smallimage 50.56.12.232
```

After the command is complete, you can confirm that the IP address has been associated with the **nova floating-ip-list** and **nova-list** commands.

```
$ nova floating-ip-list
+-----+-----+-----+-----+
| Ip | Instance Id | Fixed Ip | Pool |
+-----+-----+-----+-----+
| 50.56.12.232 | None | None | nova |
+-----+-----+-----+-----+
$ nova list
+-----+-----+
+-----+
| Networks | ID | Name | Status |
+-----+-----+
+-----+
| 4bb825ea-ea43-4771-a574-ca86ab429dcb | tinyimage2 | ACTIVE | public=10.4.113.6; private=172.16.101.6 |
| 542235df-8ba4-4d08-90c9-b79f5a77c04f | smallimage | ACTIVE | public=10.4.113.9; private=172.16.101.9 |
+-----+-----+
```

You can now de-allocate the floating IP address, returning it to the pool so that it can be used by another tenant.

```
$ nova floating-ip-delete 50.56.12.232
```

In this example, 50.56.12.232 was the only IP address allocated to this tenant. Running `nova floating-ip-list` after the de-allocation is complete will return no results.

## Manage Security Groups

A security group is a named collection of network access rules that can be used to limit the types of traffic that have access to instances. When you spawn an instance, you can assign it to one or more groups. For each security group, the associated rules permit you to manage the allowed traffic to instances within the group. Any incoming traffic which is not matched by a rule is denied by default. At any time, it is possible to add or remove rules within a security group. Rules are automatically enforced as soon as they are created.

Before you begin, use `nova secgroup-list` to view the available security groups (specify `--all-tenants` if you are a cloud administrator wanting to view all tenants' groups) . You can also view the rules for a security group with `nova secgroup-list-rules`.

```
$ nova secgroup-list
+-----+
| Name | Description |
+-----+
| default | default |
+-----+

$ nova secgroup-list-rules default
+-----+-----+-----+-----+
| IP Protocol | From Port | To Port | IP Range | Source Group |
+-----+-----+-----+-----+
| tcp | 80 | 80 | 0.0.0.0/0 | |
+-----+-----+-----+-----+
```

In this example, the default security group has been modified to allow HTTP traffic on the instance by permitting TCP traffic on Port 80.

## Add or delete a security group

Security groups can be added with **nova secgroup-create**.

The following example shows the creation of the security group `secure1`. After the group is created, it can be viewed in the security group list.

```
$ nova secgroup-create secure1 "Test security group"
+-----+
| Name | Description |
+-----+
| secure1 | Test security group |
+-----+

$ nova secgroup-list
+-----+
| Name | Description |
+-----+
| default | default |
| secure1 | Test security group |
+-----+
```

Security groups can be deleted with **nova secgroup-delete**. The default security group cannot be deleted. The default security group contains these initial settings:

- All the traffic originated by the instances (outbound traffic) is allowed
- All the traffic destined to instances (inbound traffic) is denied
- All the instances inside the group are allowed to talk to each other



### Note

You can add extra rules into the default security group for handling the egress traffic. Rules are ingress only at this time.

In the following example, the group `secure1` is deleted. When you view the security group list, it no longer appears.

```
$ nova secgroup-delete secure1
$ nova secgroup-list
+-----+
| Name | Description |
+-----+
| default | default |
+-----+
```

## Modify security group rules

The security group rules control the incoming traffic that is allowed to the instances in the group, while all outbound traffic is automatically allowed.



## Note

It is not possible to change the default outbound behaviour.

Every security group rule is a policy which allows you to specify inbound connections that are allowed to access the instance, by source address, destination port and IP protocol, (TCP, UDP or ICMP). Currently, ipv6 and other protocols cannot be managed with the security rules, making them permitted by default. To manage such, you can deploy a firewall in front of your OpenStack cloud to control other types of traffic. The command requires the following arguments for both TCP and UDP rules :

- <secgroup> ID of security group.
- <ip\_proto> IP protocol (icmp, tcp, udp).
- <from\_port> Port at start of range.
- <to\_port> Port at end of range.
- <cidr> CIDR for address range.

For ICMP rules, instead of specifying a begin and end port, you specify the allowed ICMP code and ICMP type:

- <secgroup> ID of security group.
- <ip\_proto> IP protocol (with icmp specified).
- <ICMP\_code> The ICMP code.
- <ICMP\_type> The ICMP type.
- <cidr> CIDR for the source address range.



## Note

Entering "-1" for both code and type indicates that all ICMP codes and types should be allowed.



## The CIDR notation

That notation allows you to specify a base IP address and a suffix that designates the number of significant bits in the IP address used to identify the network. For example, by specifying a 88.170.60.32/27, you specify 88.170.60.32 as the **base IP** and 27 as the **suffix**. Since you use an IPV4 format, there are only 5 bits available for the host part (32 minus 27). The 0.0.0.0/0 notation means you allow the entire IPV4 range, meaning allowing all addresses.

For example, in order to allow any IP address to access to a web server running on one of your instance inside the default security group:

```
$ nova secgroup-add-rule default tcp 80 80 0.0.0.0/0
+-----+-----+-----+-----+
| IP Protocol | From Port | To Port | IP Range | Source Group |
+-----+-----+-----+-----+
|     tcp     |      80    |      80    | 0.0.0.0/0 |             |
+-----+-----+-----+-----+
```

In order to allow any IP address to ping an instance inside the default security group (Code 0, Type 8 for the ECHO request.):

```
$ nova secgroup-add-rule default icmp 0 8 0.0.0.0/0
+-----+-----+-----+-----+
| IP Protocol | From Port | To Port | IP Range | Source Group |
+-----+-----+-----+-----+
|     icmp    |      0     |      8     | 0.0.0.0/0 |             |
+-----+-----+-----+-----+
```

```
$ nova secgroup-list-rules default
+-----+-----+-----+-----+
| IP Protocol | From Port | To Port | IP Range | Source Group |
+-----+-----+-----+-----+
|     tcp     |      80    |      80    | 0.0.0.0/0 |             |
|     icmp    |      0     |      8     | 0.0.0.0/0 |             |
+-----+-----+-----+-----+
```

In order to delete a rule, you need to specify the exact same arguments you used to create it:

- <secgroup> ID of security group.
- <ip\_proto> IP protocol (icmp, tcp, udp).
- <from\_port> Port at start of range.
- <to\_port> Port at end of range.
- <cidr> CIDR for address range.

```
$ nova secgroup-delete-rule default tcp 80 80 0.0.0.0/0
```

## 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'.
```

## Accessing running instances

The most common access method for running instances is probably ssh, but this requires you have setup your instance with [ssh keys](#) and you have arranged for it to be running ssh with a [public ip](#) and opened the ssh port in your [security group](#) configuration. If you haven't done this or you are trying to debug a problem image OpenStack can be configured to provide a VNC console, be aware that VNC is an unencrypted protocol so you should be cautious what you type across that link. See the [Getting Started With VNC Proxy](#) section for details on how to configure and connect to this service.

## Stop and Start an Instance

There are two methods for stopping and starting an instance:

- **nova pause / nova unpause**
- **nova suspend / nova resume**

## Pause and Unpause

**nova pause** stores the state of the VM in RAM. A paused instance continues to run, albeit in a "frozen" state.

## Suspend and Resume

**nova suspend** initiates a hypervisor-level suspend operation. Suspending an instance stores the state of the VM on disk; all memory is written to disk and the virtual machine is stopped. Suspending an instance is thus similar to placing a device in hibernation, and makes memory and vCPUs available. Administrators may want to suspend an instance for system maintenance, or if the instance is not frequently used.

## Change Server Configuration

After you have created a server, you may need to increase its size, change the image used to build it, or perform other configuration changes.

## Commands Used

This process uses the following commands:

- **nova resize\***
- **nova rebuild**

## Increase or Decrease Server Size

Server size is changed by applying a different flavor to the server. Before you begin, use `nova flavor-list` to review the flavors available to you.

\$ nova flavor-list								
ID	Name	Memory_MB	Disk	Ephemeral	Swap	VCPUs	RXTX_Factor	
1	m1.tiny	512	0	0		1	1.0	
2	m1.small	2048	10	20		1	1.0	
3	m1.medium	4096	10	40		2	1.0	
4	m1.large	8192	10	80		4	1.0	
5	m1.xlarge	16384	10	160		8	1.0	

In this example, we'll take a server originally configured with the `m1.tiny` flavor and resize it to `m1.small`.

\$ nova show acdfb2c4-38e6-49a9-ae1c-50182fc47e35	
Property	Value
OS-DCF:diskConfig	MANUAL
OS-EXT-STS:power_state	1
OS-EXT-STS:task_state	None
OS-EXT-STS:vm_state	active
accessIPv4	
accessIPv6	
config_drive	
created	2012-05-09T15:47:48Z
flavor	m1.tiny
hostId	de0c201e62be88c61aeb52f51d91e147acf6cf2012bb57892e528487

	id		acdfb2c4-38e6-49a9-a1c-50182fc47e35
	image		maverick-image
	key_name		
	metadata		{}
	name		resize-demo
	private network		172.16.101.6
	progress		0
	public network		10.4.113.6
	status		ACTIVE
	tenant_id		e830c2fbb7aa4586adf16d61c9b7e482
	updated		2012-05-09T15:47:59Z
	user_id		de3f4e99637743c7b6d27faca4b800a9
+-----			
+-----			

Use the `resize` command with the server's ID (`6beefcf7-9de6-48b3-9ba9-e11b343189b3`) and the ID of the desired flavor (2):

```
$ nova resize 6beefcf7-9de6-48b3-9ba9-e11b343189b3 2
```

While the server is rebuilding, its status will be displayed as RESIZING.

\$ nova list			
ID	Name	Status	Networks
970e4ca0-f9b7-4c44-80ed-bf0152c96ae1	resize-demo	RESIZE	private=172.16.101.6, public=10.4.113.6

When the `resize` operation is completed, the status displayed is `VERIFY_RESIZE`. This prompts the user to verify that the operation has been successful; to confirm:

```
$ nova resize-confirmed 6beefcf7-9de6-48b3-9ba9-e11b343189b3
```

However, if the operation has not worked as expected, you can revert it by doing:

```
$ nova resize-revert 6beefcf7-9de6-48b3-9ba9-e11b343189b3
```

In both cases, the server status should go back to ACTIVE.

## Instance evacuation

As cloud administrator, while you are managing your cloud, you may get to the point where one of the cloud compute nodes fails. For example, due to hardware malfunction. At that point you may use server evacuation in order to make managed instances available again.

### Before Evacuation

With the information about instance configuration, like if it is running on shared storage, you can choose the required evacuation parameters for your case. Use the **nova host-list** command to list the hosts and find new host for the evacuated instance. In order to preserve user data on server disk, target host has to have preconfigured shared storage with down host. As well, you have to validate that the current vm host is down. Otherwise the evacuation will fail with error.

### To evacuate your server without shared storage:

**nova evacuate** performs an instance evacuation from down host to specified host. The instance will be booted from a new disk, but will preserve the configuration, e.g. id, name, uid, ip...etc. New instance password can be passed to the command using the **--password <pwd>** option. If not given it will be generated and printed after the command finishes successfully.

```
$nova evacuate evacuated_server_name host_b
```

The command returns a new server password.

Property	Value
adminPass	kRAJpErnT4xZ

### Evacuate server to specified host and preserve user data

In order to preserve the user disk data on the evacuated server the OpenStack Compute should be deployed with shared filesystem. Refer to the shared storage section in the [Configure migrations guide](#) in order to configure your system. In this scenario the password will remain unchanged.

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

## Terminate an Instance

When you no longer need an instance, use the **nova delete** command to terminate it. You can use the instance name or the ID string. You will not receive a notification indicating that the instance has been deleted, but if you run the **nova list** command, the instance will no longer appear in the list.

In this example, we will delete the instance `tinyimage`, which is experiencing an error condition.

```
$ nova list
+-----+-----+
+-----+
|           ID      |     Name    | Status |
| Networks   |               |
+-----+-----+
+-----+
| 30ed8924-f1a5-49c1-8944-b881446a6a51 | tinyimage  | ERROR   | public=10.4.
113.11; private=172.16.101.11 |
| 4bb825ea-ea43-4771-a574-ca86ab429dc | tinyimage2 | ACTIVE  | public=10.4.
113.6; private=172.16.101.6 |
| 542235df-8ba4-4d08-90c9-b79f5a77c04f | smallimage | ACTIVE  | public=10.4.
113.9; private=172.16.101.9 |
+-----+-----+
+-----+
$ nova delete tinyimage
$ nova list
+-----+-----+
+-----+
|           ID      |     Name    | Status |
| Networks   |               |
+-----+-----+
+-----+
| 4bb825ea-ea43-4771-a574-ca86ab429dc | tinyimage2 | ACTIVE  | public=10.4.
113.6; private=172.16.101.6 |
| 542235df-8ba4-4d08-90c9-b79f5a77c04f | smallimage | ACTIVE  | public=10.4.
113.9; private=172.16.101.9 |
+-----+-----+
+-----+
```

# 9. Hypervisors

## Table of Contents

Selecting a Hypervisor .....	173
Hypervisor Configuration Basics .....	174
KVM .....	176
QEMU .....	180
Xen, XenAPI, XenServer and XCP .....	181
LXC (Linux containers) .....	189
VMware vSphere .....	189
PowerVM .....	192
Hyper-V Virtualization Platform .....	192
Bare Metal Driver .....	198
Nova Compute Fibre Channel Support .....	199

This section assumes you have a working installation of OpenStack Compute and want to select a particular hypervisor or run with multiple hypervisors. Before you try to get a VM running within OpenStack Compute, be sure you have installed a hypervisor and used the hypervisor's documentation to run a test VM and get it working.

## Selecting a Hypervisor

OpenStack Compute supports many hypervisors, an array of which must provide a bit of difficulty in selecting a hypervisor unless you are already familiar with one. Most installations only use a single hypervisor, however it is possible to use the [ComputeFilter](#) and [ImagePropertiesFilter](#) to allow scheduling to different hypervisors within the same installation. The following links provide additional information for choosing a hypervisor. Refer to <http://wiki.openstack.org/HypervisorSupportMatrix> for a detailed list of features and support across the hypervisors.

Here is a list of the supported hypervisors with links to a relevant web site for configuration and use:

- [KVM](#) - Kernel-based Virtual Machine. The virtual disk formats that it supports it inherits from QEMU since it uses a modified QEMU program to launch the virtual machine. The supported formats include raw images, the qcow2, and VMware formats.
- [LXC](#) - Linux Containers (through libvirt), use to run Linux-based virtual machines.
- [QEMU](#) - Quick EMULATOR, generally only used for development purposes.
- [UML](#) - User Mode Linux, generally only used for development purposes.
- [VMWare vSphere](#) 4.1 update 1 and newer, runs VMWare-based Linux and Windows images through a connection with a vCenter server or directly with an ESXi host.
- [Xen](#) - XenServer, Xen Cloud Platform (XCP), use to run Linux or Windows virtual machines. You must install the nova-compute service in a para-virtualized VM.

- **PowerVM** - Server virtualization with IBM PowerVM, used to run AIX, IBM i and Linux environments on IBM POWER technology.
  - **Hyper-V** - Server virtualization with Microsoft's Hyper-V, used to run Windows, Linux, and FreeBSD virtual machines. Runs nova-compute natively on the Windows virtualization platform.
  - **Bare Metal** - Not a hypervisor in the traditional sense, this driver provisions physical hardware via pluggable sub-drivers (eg. PXE for image deployment, and IPMI for power management).

# Hypervisor Configuration Basics

The node where the nova-compute service is installed and running is the machine that runs all the virtual machines, referred to as the compute node in this guide.

By default, the selected hypervisor is KVM. To change to another hypervisor, change the libvirt\_type option in nova.conf and restart the nova-compute service.

Here are the nova.conf options that are used to configure the compute node.

**Table 9.1. Description of nova.conf file configuration options for hypervisors**

Configuration option=Default value	(Type) Description
block_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PERSIST_DEST,VIR_MIGRATE_BLOCKED_BY_SNAPSHOT,VIR_MIGRATE_BLOCKED_BY_SNAPSHOT_INACTIVE,VIR_MIGRATE_BLOCKED_BY_SNAPSHOT_INACTIVE,NON_SHARED_INC	(StringOpt) Define block migration flags
checksum_base_images=false	(BoolOpt) Used as an additional check to detect if cached images have become corrupted. If true, the compute service will write checksums for image files in the /var/lib/nova/instances/_base directory to disk, and do periodic checks to verify that this checksum is valid. If the checksum fails to validate, the failure is recorded to the log as an error, but no other action is taken: it is assumed that an operator will monitor the logs and take appropriate action. If false, image hashes are not verified.
hyperv_attaching_volume_retry_count=10	(IntOpt) Number of times to retry attaching to a volume when using the Hyper-V hypervisor
hyperv_wait_between_attach_retry=5	(IntOpt) To be written: found in /nova/virt/hyperv/volumeops.py
libvirt_cpu_mode=<None>	(StrOpt) Configures the guest CPU model exposed to the hypervisor. Valid options are: custom, host-model, host-passthrough, none. If the hypervisor is KVM or QEMU, the default value is host-model, otherwise the default value is none.
libvirt_cpu_model=<None>	(StrOpt) Specify the guest CPU model exposed to the hypervisor. This configuration option is only applicable if libvirt_cpu_mode is set to custom. Valid options: one of the named models specified in /usr/share/libvirt/cpu_map.xml, e.g.: Westmere, Nehalem, Opteron_G3.
libvirt_disk_prefix=<None>	(StrOpt) Override the default disk prefix for the devices attached to a server, which is dependent on libvirt_type. (valid options are: sd, xvd, uvd, vd)
libvirt_inject_key=true	(BoolOpt) Inject the ssh public key at boot time
libvirt_inject_partition=1	(IntOpt) The partition to inject to : -2 => disable, -1 => inspect (libguestfs only), 0 => not partitioned, >0 => partition number'

Configuration option=Default value	(Type) Description
libvirt_images_type=default	(StrOpt) Instance ephemeral storage backend format. Acceptable values are: raw, qcow2, lvm, default. If default is specified, then use_cow_images flag is used instead of this one. Please note, that current snapshot mechanism in OpenStack Compute works only with instances backed with Qcow2 images.
libvirt_images_volume_group=None	(StrOpt) LVM Volume Group that is used for instance ephemerals, when you specify libvirt_images_type=lvm.
libvirt_inject_password=false	(BoolOpt) Inject the admin password at boot time, without an agent.
libvirt_lvm_snapshot_size=1000	(IntOpt) The amount of storage (in megabytes) to allocate for LVM snapshot copy-on-write blocks.
libvirt_nonblocking=true	(BoolOpt) Use a separated OS thread pool to realize non-blocking libvirt calls
libvirt_snapshots_directory=\$instances_path/snapshots	(StrOpt) Location where libvirt driver will store snapshots before uploading them to image service
libvirt_snapshot_compression=False	(BoolOpt) Compresses snapshot images when possible. This currently applies exclusively to qcow2 images.
libvirt_sparse_logical_volumes=false	(BoolOpt) Create sparse (not fully allocated) LVM volumes for instance ephemerals if you use LVM backend for them.
libvirt_type=kvm	(StrOpt) Libvirt domain type (valid options are: kvm, lxc, qemu, uml, xen)
libvirt_uri=	(StrOpt) Override the default libvirt URI (which is dependent on libvirt_type)
libvirt_vif_driver=nova.virt.libvirt.vif.LibvirtGenericVifDriver	(StrOpt) The libvirt VIF driver to configure the VIFs.
libvirt_volume_drivers="iscsi=nova.virt.libvirt.volume.LibvirtISCSIDriver, rbd=nova.virt.libvirt.volume.LibvirtRBDVolumeDriver, fake=nova.virt.libvirt.volume.LibvirtFakeVolumeDriver, rbd=nova.virt.libvirt.volume.LibvirtNetVolumeDriver, sheepdog=nova.virt.libvirt.volume.LibvirtNetVolumeDriver, glusterfs=nova.virt.libvirt.volume.LibvirtGlusterfsVolumeDriver"	(StrOpt) libvirt handlers for remote volumes.
libvirt_wait_soft_reboot_seconds=120	(IntOpt) Number of seconds to wait for instance to shutdown after soft reboot request is made. We fall back to hard reboot if instance does not shutdown within this window.
limit_cpu_features=false	(BoolOpt) Used by Hyper-V
remove_unused_base_images=true	(BoolOpt) Indicates whether unused base images should be removed
remove_unused_kernels=false	(BoolOpt) Should unused kernel images be removed? If unused images should be removed set to true, if not, set to false. This option is only safe to set to true if all compute nodes have been updated to support this option so that older image cache managers on remote compute nodes are prevented from cleaning up kernels because they appear unused. This will be enabled by default in a future release.
remove_unused_original_minimum_age_seconds=86400	(IntOpt) Unused unresized base images younger than this will not be removed
remove_unused_resized_minimum_age_seconds=3600	(IntOpt) Unused resized base images younger than this will not be removed
rescue_image_id=<None>	(StrOpt) Rescue ami image
rescue_kernel_id=<None>	(StrOpt) Rescue aki image

Configuration option=Default value	(Type) Description
rescue_ramdisk_id=<None>	(StrOpt) Rescue ari image
snapshot_image_format=<None>	(StrOpt) Snapshot image format (valid options are : raw, qcow2, vmdk, vdi). Defaults to same as source image
use_usb_tablet=true	(BoolOpt) Sync virtual and real mouse cursors in Windows VMs
libvirt integration	
libvirt_ovs_bridge=br-int	(StrOpt) Name of Integration Bridge used by Open vSwitch
libvirt_use_virtio_for_bridges=false	(BoolOpt) Use virtio for bridge interfaces
VMWare integration	
vmwareapi_wsdl_loc=<None>	(StrOpt) VIM Service WSDL Location e.g http://<server>/vimService.wsdl, due to a bug in vSphere ESX 4.1 default wsdl.
vmware_vif_driver=nova.virt.vmwareapi.vif.VMWareVlanBridgeDriver	(TypeOpt) The VMWare VIF driver to configure the VIFs.
vmwareapi_api_retry_count=10	(FloatOpt) The number of times we retry on failures, e.g., socket error, etc. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_ip=<None>	(StrOpt) URL for connection to VMWare ESX host. Required if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_password=<None>	(StrOpt) Password for connection to VMWare ESX host. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_host_username=<None>	(StrOpt) Username for connection to VMWare ESX host. Used only if compute_driver is vmwareapi.VMwareESXDriver.
vmwareapi_task_poll_interval=5.0	(FloatOpt) The interval used for polling of remote tasks. Used only if compute_driver is vmwareapi.VMwareESXDriver,
vmwareapi_vlan_interface=vmnic0	(StrOpt) Physical ethernet adapter name for vlan networking
powervm_mgr_type=ivm	(StrOpt) PowerVM system manager type (ivm, hmc)
powervm_mgr=<None>	(StrOpt) PowerVM manager host or ip
powervm_vios=powervm_mgr	(StrOpt) PowerVM VIOS host or ip if different from manager
powervm_mgr_user=<None>	(StrOpt) PowerVM manager user name
powervm_mgr_passwd=<None>	(StrOpt) PowerVM manager user password
powervm_img_remote_path=<None>	(StrOpt) PowerVM image remote path. Used to copy and store images from Glance on the PowerVM VIOS LPAR.
powervm_img_local_path=<None>	(StrOpt) Local directory on the compute host to download glance images to.

## KVM

KVM is configured as the default hypervisor for Compute.



### Note

There are several sections about hypervisor selection in this document. If you are reading this document linearly, you do not want to load the KVM module prior to installing nova-compute. The nova-compute service depends on qemu-

kvm which installs /lib/udev/rules.d/45-qemu-kvm.rules, which sets the correct permissions on the /dev/kvm device node.

To enable KVM explicitly, add the following configuration options /etc/nova/nova.conf:

```
compute_driver=libvirt.LibvirtDriver
libvirt_type=kvm
```

The KVM hypervisor supports the following virtual machine image formats:

- Raw
- QEMU Copy-on-write (qcow2)
- VMWare virtual machine disk format (vmdk)

The rest of this section describes how to enable KVM on your system. You may also wish to consult distribution-specific documentation:

- [Fedora: Getting started with virtualization](#) from the Fedora project wiki.
- [Ubuntu: KVM/Installation](#) from the Community Ubuntu documentation.
- [Debian: Virtualization with KVM](#) from the Debian handbook.
- [RHEL: Installing virtualization packages on an existing Red Hat Enterprise Linux system](#) from the Red Hat Enterprise Linux Virtualization Host Configuration and Guest Installation Guide.
- [openSUSE: Installing KVM](#) from the openSUSE Virtualization with KVM manual.
- [SLES: Installing KVM](#) from the SUSE Linux Enterprise Server Virtualization with KVM manual.

## Checking for hardware virtualization support

The processors of your compute host need to support virtualization technology (VT) to use KVM.

If you are running on Ubuntu, install the cpu package and use the **kvm-ok** command to check if your processor has VT support, it is enabled in the BIOS, and KVM is installed properly, as root:

```
# apt-get install cpu
# kvm-ok
```

If KVM is enabled, the output should look something like:

```
INFO: /dev/kvm exists
KVM acceleration can be used
```

If KVM is not enabled, the output should look something like:

```
INFO: Your CPU does not support KVM extensions
```

```
KVM acceleration can NOT be used
```

In the case that KVM acceleration is not supported, Compute should be configured to use a different hypervisor, such as [QEMU](#) or [Xen](#).

On distributions that don't have **kvm-ok**, you can check if your processor has VT support by looking at the processor flags in the `/proc/cpuinfo` file. For Intel processors, look for the `vmx` flag, and for AMD processors, look for the `svm` flag. A simple way to check is to run the following command and see if there is any output:

```
$ egrep '(vmx|svm)' --color=always /proc/cpuinfo
```

Some systems require that you enable VT support in the system BIOS. If you believe your processor supports hardware acceleration but the above command produced no output, you may need to reboot your machine, enter the system BIOS, and enable the VT option.

## Enabling KVM

KVM requires the `kvm` and either `kvm-intel` or `kvm-amd` modules to be loaded. This may have been configured automatically on your distribution when KVM is installed.

You can check that they have been loaded using `lsmod`, as follows, with expected output for Intel-based processors:

```
$ lsmod | grep kvm
kvm_intel           137721   9
kvm                 415459   1 kvm_intel
```

The following sections describe how to load the kernel modules for Intel-based and AMD-based processors if they were not loaded automatically by your distribution's KVM installation process.

## Intel-based processors

If your compute host is Intel-based, run the following as root to load the kernel modules:

```
# modprobe kvm
# modprobe kvm-intel
```

Add the following lines to `/etc/modules` so that these modules will load on reboot:

```
kvm
kvm-intel
```

## AMD-based processors

If your compute host is AMD-based, run the following as root to load the kernel modules:

```
# modprobe kvm
# modprobe kvm-amd
```

Add the following lines to `/etc/modules` so that these modules will load on reboot:

```
kvm
kvm-amd
```

## Specifying the CPU model of KVM guests

The Compute service allows you to control the guest CPU model that is exposed to KVM virtual machines. Use cases include:

- To maximise performance of virtual machines by exposing new host CPU features to the guest
- To ensure a consistent default CPU across all machines, removing reliance of variable QEMU defaults

In libvirt, the CPU is specified by providing a base CPU model name (which is a shorthand for a set of feature flags), a set of additional feature flags, and the topology (sockets/cores/threads). The libvirt KVM driver provides a number of standard CPU model names. Examples of model names include:

```
"486", "pentium", "pentium2", "pentiumpro", "coreduo", "n270",
"pentiumpro", "qemu32", "kvm32", "cpu64-rhel5", "cpu64-rhel5",
"kvm64", "pentiumpro", "Conroe", "Penryn", "Nehalem", "Westmere",
"pentiumpro", "cpu64-rhel5", "cpu64-rhel5", "Opteron_G1",
"Opteron_G2", "Opteron_G3", "Opteron_G4"
```

These models are defined in the file `/usr/share/libvirt/cpu_map.xml`. Check this file to determine which models are supported by your local installation.

There are two Compute configuration options that determine the type of CPU model exposed to the hypervisor when using KVM, `libvirt_cpu_mode` and `libvirt_cpu_model`.

The `libvirt_cpu_mode` option can take one of four values: `none`, `host-passthrough`, `host-model` and `custom`.

### Host model (default for KVM & QEMU)

If your `nova.conf` contains `libvirt_cpu_mode=host-model`, libvirt will identify the CPU model in `/usr/share/libvirt/cpu_map.xml` which most closely matches the host, and then request additional CPU flags to complete the match. This should give close to maximum functionality/performance, which maintaining good reliability/compatibility if the guest is migrated to another host with slightly different host CPUs.

### Host passthrough

If your `nova.conf` contains `libvirt_cpu_mode=host-passthrough`, libvirt will tell KVM to passthrough the host CPU with no modifications. The difference to host-model, instead of just matching feature flags, every last detail of the host CPU is matched. This gives absolutely best performance, and can be important to some apps which check low level CPU details, but it comes at a cost with respect to migration: the guest can only be migrated to an exactly matching host CPU.

### Custom

If your `nova.conf` contains `libvirt_cpu_mode=custom`, you can explicitly specify one of the supported named model using the `libvirt_cpu_model` configuration option. For

example, to configure the KVM guests to expose Nehalem CPUs, your `nova.conf` should contain:

```
libvirt_cpu_mode=custom
libvirt_cpu_model=Nehalem
```

## None (default for all libvirt-driven hypervisors other than KVM & QEMU)

If your `nova.conf` contains `libvirt_cpu_mode=none`, then libvirt will not specify any CPU model at all. It will leave it up to the hypervisor to choose the default model. This setting is equivalent to the Compute service behavior prior to the Folsom release.

## Troubleshooting

Trying to launch a new virtual machine instance fails with the `ERROR` state, and the following error appears in `/var/log/nova/nova-compute.log`

```
libvirtError: internal error no supported architecture for os type 'hvm'
```

This is a symptom that the KVM kernel modules have not been loaded.

If you cannot start VMs after installation without rebooting, it's possible the permissions are not correct. This can happen if you load the KVM module before you've installed `nova-compute`. To check the permissions, run `ls -l /dev/kvm` to see whether the group is set to `kvm`. If not, run `sudo udevadm trigger`.

## QEMU

From the perspective of the Compute service, the QEMU hypervisor is very similar to the KVM hypervisor. Both are controlled through libvirt, both support the same feature set, and all virtual machine images that are compatible with KVM are also compatible with QEMU. The main difference is that QEMU does not support native virtualization. Consequently, QEMU has worse performance than KVM and is a poor choice for a production deployment.

The typical uses cases for QEMU are

- Running on older hardware that lacks virtualization support.
- Running the Compute service inside of a virtual machine for development or testing purposes, where the hypervisor does not support native virtualization for guests.

KVM requires hardware support for acceleration. If hardware support is not available (e.g., if you are running Compute inside of a VM and the hypervisor does not expose the required hardware support), you can use QEMU instead. KVM and QEMU have the same level of support in OpenStack, but KVM will provide better performance. To enable QEMU:

```
compute_driver=libvirt.LibvirtDriver
libvirt_type=qemu
```

For some operations you may also have to install the `guestmount` utility:

```
$> sudo apt-get install guestmount
```

```
$> sudo yum install libguestfs-tools
```

The QEMU hypervisor supports the following virtual machine image formats:

- Raw
- QEMU Copy-on-write (qcow2)
- VMWare virtual machine disk format (vmdk)

## Tips and fixes for QEMU on RHEL

If you are testing OpenStack in a virtual machine, you need to configure nova to use qemu without KVM and hardware virtualization. The second command relaxes SELinux rules to allow this mode of operation ([https://bugzilla.redhat.com/show\\_bug.cgi?id=753589](https://bugzilla.redhat.com/show_bug.cgi?id=753589)) The last 2 commands here work around a libvirt issue fixed in RHEL 6.4. Note nested virtualization will be the much slower TCG variety, and you should provide lots of memory to the top level guest, as the OpenStack-created guests default to 2GM RAM with no overcommit.



### Note

The second command, `setsebool`, may take a while.

```
$> sudo openstack-config --set /etc/nova/nova.conf DEFAULT libvirt_type qemu
$> setsebool -P virt_use_execmem on
$> sudo ln -s /usr/libexec/qemu-kvm /usr/bin/qemu-system-x86_64
$> sudo service libvirtd restart
```

## Xen, XenAPI, XenServer and XCP

The recommended way to use Xen with OpenStack is through the XenAPI driver. To enable the XenAPI driver, add the following configuration options `/etc/nova/nova.conf` and restart the `nova-compute` service:

```
compute_driver=xenapi.XenAPIDriver
xenapi_connection_url=http://your_xenapi_management_ip_address
xenapi_connection_username=root
xenapi_connection_password=your_password
```

The above connection details are used by the OpenStack Compute service to contact your hypervisor and are the same details you use to connect XenCenter, the XenServer management console, to your XenServer or XCP box. Note these settings are generally unique to each hypervisor host as the use of the host internal management network IP address (169.254.0.1) will cause features such as live-migration to break.

OpenStack with XenAPI supports the following virtual machine image formats:

- Raw
- VHD (in a gzipped tarball)

It is possible to manage Xen using libvirt. This would be necessary for any Xen-based system that isn't using the XCP toolstack, such as SUSE Linux or Oracle Linux. Unfortunately, this is not well-tested or supported. To experiment using Xen through libvirt add the following configuration options `/etc/nova/nova.conf`:

```
compute_driver=libvirt.LibvirtDriver
libvirt_type=xen
```

The rest of this section describes Xen, XCP, and XenServer, the differences between them, and how to use them with OpenStack. Xen's architecture is different from KVM's in important ways, and we discuss those differences and when each might make sense in your OpenStack cloud.

## Xen terminology

**Xen** is a hypervisor. It provides the fundamental isolation between virtual machines. Xen is open source (GPLv2) and is managed by Xen.org, an cross-industry organization.

Xen is a component of many different products and projects. The hypervisor itself is very similar across all these projects, but the way that it is managed can be different, which can cause confusion if you're not clear which tool stack you are using. Make sure you know what tool stack you want before you get started.

**Xen Cloud Platform (XCP)** is an open source (GPLv2) tool stack for Xen. It is designed specifically as platform for enterprise and cloud computing, and is well integrated with OpenStack. XCP is available both as a binary distribution, installed from an iso, and from Linux distributions, such as [xcp-xapi](#) in Ubuntu. The current versions of XCP available in Linux distributions do not yet include all the features available in the binary distribution of XCP.

**Citrix XenServer** is a commercial product. It is based on XCP, and exposes the same tool stack and management API. As an analogy, think of XenServer being based on XCP in the way that Red Hat Enterprise Linux is based on Fedora. XenServer has a free version (which is very similar to XCP) and paid-for versions with additional features enabled. Citrix provides support for XenServer, but as of July 2012, they do not provide any support for XCP. For a comparison between these products see the [XCP Feature Matrix](#).

Both XenServer and XCP include Xen, Linux, and the primary control daemon known as [xapi](#).

The API shared between XCP and XenServer is called **XenAPI**. OpenStack usually refers to XenAPI, to indicate that the integration works equally well on XCP and XenServer. Sometimes, a careless person will refer to XenServer specifically, but you can be reasonably confident that anything that works on XenServer will also work on the latest version of XCP. Read the [XenAPI Object Model Overview](#) for definitions of XenAPI specific terms such as SR, VDI, VIF and PIF.

## Privileged and unprivileged domains

A Xen host will run a number of virtual machines, VMs, or domains (the terms are synonymous on Xen). One of these is in charge of running the rest of the system, and is known as "domain 0", or "dom0". It is the first domain to boot after Xen, and owns the storage and networking hardware, the device drivers, and the primary control software.

Any other VM is unprivileged, and are known as a "domU" or "guest". All customer VMs are unprivileged of course, but you should note that on Xen the OpenStack control software (nova-compute) also runs in a domU. This gives a level of security isolation between the privileged system software and the OpenStack software (much of which is customer-facing). This architecture is described in more detail later.

There is an ongoing project to split domain 0 into multiple privileged domains known as **driver domains** and **stub domains**. This would give even better separation between critical components. This technology is what powers Citrix XenClient RT, and is likely to be added into XCP in the next few years. However, the current architecture just has three levels of separation: dom0, the OpenStack domU, and the completely unprivileged customer VMs.

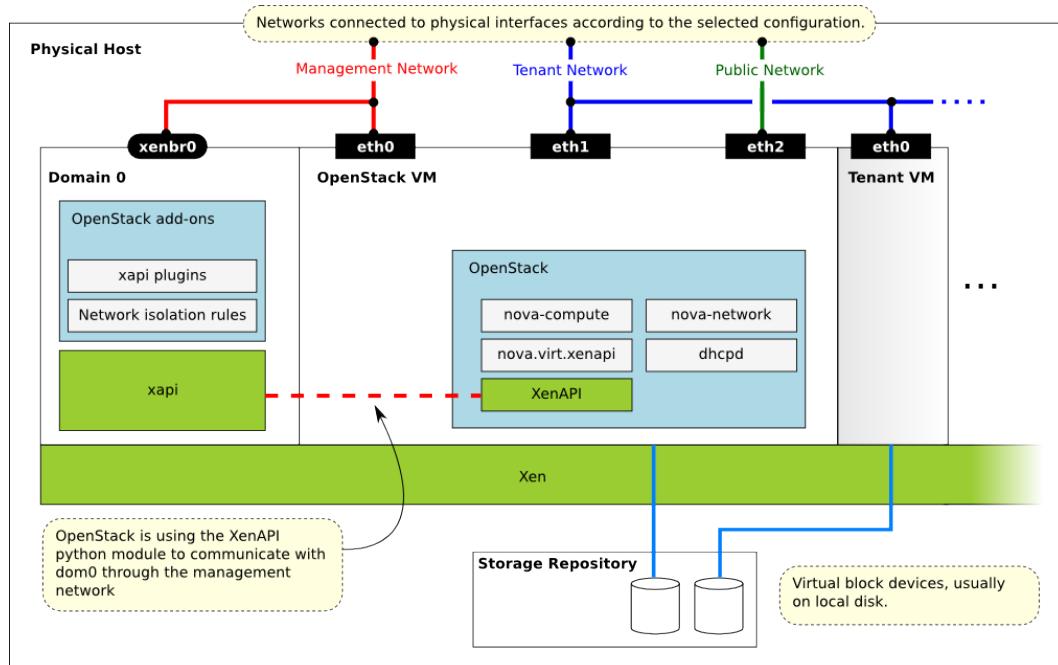
## Paravirtualized versus hardware virtualized domains

A Xen virtual machine can be **paravirtualized (PV)** or **hardware virtualized (HVM)**. This refers to the interaction between Xen, domain 0, and the guest VM's kernel. PV guests are aware of the fact that they are virtualized and will co-operate with Xen and domain 0; this gives them better performance characteristics. HVM guests are not aware of their environment, and the hardware has to pretend that they are running on an unvirtualized machine. HVM guests have the advantage that there is no need to modify the guest operating system, which is essential when running Windows.

In OpenStack, customer VMs may run in either PV or HVM mode. However, the OpenStack domU (that's the one running nova-compute) **must** be running in PV mode.

## XenAPI deployment architecture

When you deploy OpenStack on XCP or XenServer you will get something similar to this:



Key things to note:

- The hypervisor: Xen
- Domain 0: runs xapi and some small pieces from OpenStack (some xapi plugins and network isolation rules). The majority of this is provided by XenServer or XCP (or yourself using Kronos).
- OpenStack VM: The nova-compute code runs in a paravirtualized virtual machine, running on the host under management. Each host runs a local instance of nova-compute. It will often also be running nova-network (depending on your network mode). In this case, nova-network is managing the addresses given to the tenant VMs through DHCP.
- Nova uses the XenAPI Python library to talk to xapi, and it uses the Management Network to reach from the domU to dom0 without leaving the host.

Some notes on the networking:

- The above diagram assumes FlatDHCP networking (the DevStack default).
- There are three main OpenStack Networks:
  - Management network - RabbitMQ, MySQL, etc. Please note, that the VM images are downloaded by the xenapi plugins, so please make sure, that the images could be downloaded through the management network. It usually means, binding those services to the management interface.
  - Tenant network - controlled by nova-network. The parameters of this network depends on the networking model selected (Flat, Flat DHCP, VLAN)
  - Public network - floating IPs, public API endpoints.
- The networks shown here need to be connected to the corresponding physical networks within the datacenter. In the simplest case, three individual physical network cards could be used. It is also possible to use VLANs to separate these networks. Please note, that the selected configuration must be in line with the networking model selected for the cloud. (in case of VLAN networking, the physical channels have to be able to forward the tagged traffic)

## XenAPI pools

The host-aggregates feature allows you to create pools of XenServer hosts (configuring shared storage is still an out of band activity), to enable live migration when using shared storage.

## Installing XenServer and XCP

When you want to run OpenStack with XCP or XenServer, you first need to install the software on [an appropriate server](#). Please note, Xen is a type 1 hypervisor. This means when your server starts the first software that runs is Xen. This means the software you install on your compute host is XenServer or XCP, not the operating system you wish to run the OpenStack code on. The OpenStack services will run in a VM you install on top of XenServer.

Before you can install your system you must decide if you want to install Citrix XenServer (either the free edition, or one of the paid editions) or Xen Cloud Platform from Xen.org. You can download the software from the following locations:

- <http://www.citrix.com/XenServer/download>
- <http://www.xen.org/download/xcp/index.html>

When installing many servers, you may find it easier to perform [PXE boot installations of XenServer or XCP](#). You can also package up any post install changes you wish to make to your XenServer by [creating your own XenServer supplemental pack](#).

It is also possible to get XCP by installing the **xcp-xenapi** package on Debian based distributions. However, this is not as mature or feature complete as above distributions. This will modify your boot loader to first boot Xen, then boot your existing OS on top of Xen as Dom0. It is in Dom0 that the xapi daemon will run. You can find more details on the Xen.org wiki: [http://wiki.xen.org/wiki/Project\\_Kronos](http://wiki.xen.org/wiki/Project_Kronos)



### Important

Ensure you are using the EXT type of storage repository (SR). Features that require access to VHD files (such as copy on write, snapshot and migration) do not work when using the LVM SR. Storage repository (SR) is a XenAPI specific term relating to the physical storage on which virtual disks are stored.

On the XenServer/XCP installation screen, this is selected by choosing "XenDesktop Optimized" option. In case you are using an answer file, make sure you use `srtype= "ext"` within the `installation` tag of the answer file.

## Post install steps

You are now ready to install OpenStack onto your XenServer system. This process involves the following steps:

- For resize and migrate functionality, please perform the changes described in the [Configuring Resize](#) section of the OpenStack Compute Administration Manual.
- Install the VIF isolation rules to help prevent mac and ip address spoofing.
- Install the XenAPI plugins - see the next section.
- In order to support AMI type images, you need to set up `/boot/guest symlink/` directory in Dom0. For detailed instructions, see next section.
- To support resize/migration, set up an ssh trust relation between your XenServer hosts, and ensure `/images` is properly set up. See next section for more details.
- Create a Paravirtualised virtual machine that can run the OpenStack compute code.
- Install and configure the nova-compute in the above virtual machine.

For further information on these steps look at how DevStack performs the last three steps when doing developer deployments. For more information on DevStack, take a look at the

[DevStack and XenServer Readme](#). More information on the first step can be found in the [XenServer mutli-tenancy protection doc](#). More information on how to install the XenAPI plugins can be found in the [XenAPI plugins Readme](#).

## Installing the XenAPI Plugins

When using Xen as the hypervisor for OpenStack Compute, you can install a Python script (usually, but it can be any executable) on the host side, and then call that through the XenAPI. These scripts are called plugins. The XenAPI plugins live in the nova code repository. These plugins have to be copied to the hypervisor's Dom0, to the appropriate directory, where xapi can find them. There are several options for the installation. The important thing is to ensure that the version of the plugins are in line with the nova installation by only installing plugins from a matching nova repository.

Manual Installation:

- Create temporary files/directories:

```
$ NOVA_ZIPBALL=$(mktemp)
$ NOVA_SOURCES=$(mktemp -d)
```

- Get the source from github. The example assumes the master branch is used, please amend the URL to match the version being used:

```
$ wget -qO "$NOVA_ZIPBALL" https://github.com/openstack/nova/archive/master.zip
$ unzip "$NOVA_ZIPBALL" -d "$NOVA_SOURCES"
```

(Alternatively) Should you wish to use the official Ubuntu packages, use the following commands to get the nova codebase:

```
$ ( cd $NOVA_SOURCES && apt-get source python-nova --download-only )
$ ( cd $NOVA_SOURCES && for ARCHIVE in *.tar.gz; do tar -xzf $ARCHIVE;
done )
```

- Copy the plugins to the hypervisor:

```
$ PLUGINPATH=$(find $NOVA_SOURCES -path '*/xapi.d/plugins' -type d -print)
$ tar -czf - -C "$PLUGINPATH" ./ | ssh root@xenserver tar -xzf - -C /etc/xapi.d/plugins/
```

- Remove the temporary files/directories:

```
$ rm "$NOVA_ZIPBALL"
$ rm -rf "$NOVA_SOURCES"
```

Packaged Installation:

Follow these steps to produce a supplemental pack from the nova sources, and package it as a XenServer Supplemental Pack.

- Create RPM packages. Given you have the nova sources (use one of the methods mentioned at Manual Installation):

```
$ cd nova/plugins/xenserver/xenapi/contrib  
$ ./build-rpm.sh
```

The above commands should leave an .rpm file in the rpmbuild/RPMS/noarch/ directory.

- Pack the RPM packages to a Supplemental Pack, using the XenServer DDK (the following command should be issued on the XenServer DDK virtual appliance, after the produced rpm file has been copied over):

```
$ /usr/bin/build-supplemental-pack.sh \  
> --output=output_directory \  
> --vendor-code=novaplugin \  
> --vendor-name=openstack \  
> --label=novaplugins \  
> --text="nova plugins" \  
> --version=0 \  
> full_path_to_rpmfile
```

The above command should produce an .iso file in the output directory specified. Copy that file to the hypervisor.

- Install the Supplemental Pack. Log in to the hypervisor, and issue:

```
# xe-install-supplemental-pack path_to_isofile
```

## Prepare for AMI Type Images

In order to support AMI type images within your OpenStack installation, a directory /boot/guest needs to be created inside Dom0. The OpenStack VM will put the kernel and ramdisk extracted from the AKI and ARI images to this location.

This directory's content will be maintained by OpenStack, and its size should not increase during normal operation, however in case of power failures or accidental shutdowns, some files might be left over. In order to prevent these files to fill up Dom0's disk, it is recommended to set up this directory as a symlink pointing to a subdirectory of the local SR.

Execute the following commands in Dom0 to achieve the above mentioned setup:

```
# LOCAL_SR=$(xe sr-list name-label="Local storage" --minimal)  
# LOCALPATH="/var/run/sr-mount/$LOCAL_SR/os-guest-kernels"  
# mkdir -p "$LOCALPATH"  
# ln -s "$LOCALPATH" /boot/guest
```

## Dom0 Modifications for Resize/Migration Support

To get resize to work with XenServer (and XCP) you need to:

- Establish a root trust between all hypervisor nodes of your deployment:

You can do so by generating an ssh key-pair (with **ssh-keygen**) and then ensuring that each of your dom0's authorized\_keys file (located in /root/.ssh/

authorized\_keys) contains the public key fingerprint (located in /root/.ssh/id\_rsa.pub).

- Provide an /images mount point to your hypervisor's dom0:

Dom0 space is a premium so creating a directory in dom0 is kind of dangerous, and almost surely bound to fail especially when resizing big servers. The least you can do is to symlink /images to your local storage SR. The instructions below work for an English-based installation of XenServer (and XCP) and in the case of ext3 based SR (with which the resize functionality is known to work correctly).

```
# LOCAL_SR=$(xe sr-list name-label="Local storage" --minimal)
# IMG_DIR="/var/run/sr-mount/$LOCAL_SR/images"
# mkdir -p "$IMG_DIR"
# ln -s "$IMG_DIR" /images
```

## Xen Boot from ISO

XenServer, through the XenAPI integration with OpenStack provides a feature to boot instances from an ISO file. In order to activate the "Boot From ISO" feature, the SR elements on XenServer host must be configured that way.

First, create an ISO-typed SR, such as an NFS ISO library, for instance. For this, using XenCenter is a simple method. You need to export an NFS volume from a remote NFS server. Make sure it is exported in read-write mode.

Second, on the compute host, find the uuid of this ISO SR and write it down.

```
# xe host-list
```

Next, locate the uuid of the NFS ISO library:

```
# xe sr-list content-type=iso
```

Set the uuid and configuration. Even if an NFS mount point isn't local storage, you must specify "local-storage-iso".

```
# xe sr-param-set uuid=[iso sr uuid] other-config:i18n-key=local-storage-iso
```

Now, make sure the host-uuid from "xe pbd-list" equals the uuid of the host you found earlier

```
# xe sr-uuid=[iso sr uuid]
```

You should now be able to add images via the OpenStack Image Registry, with disk-format=iso, and boot them in OpenStack Compute.

```
glance image-create --name=fedora_iso --disk-format=iso --container-format=bare < Fedora-16-x86_64-netinst.iso
```

## Further reading

Here are some of the resources available to learn more about Xen:

- Citrix XenServer official documentation: <http://docs.vmd.citrix.com/XenServer>.

- What is Xen? by Xen.org: <http://xen.org/files/Marketing/WhatisXen.pdf>.
- Xen Hypervisor project: <http://xen.org/products/xenhyp.html>.
- XCP project: <http://xen.org/products/cloudxen.html>.
- Further XenServer and OpenStack information: <http://wiki.openstack.org/XenServer>.

## LXC (Linux containers)

LXC (also known as Linux containers) is a virtualization technology that works at the operating system level. This is different from hardware virtualization, the approach used by other hypervisors such as KVM, Xen, and VMWare. LXC (as currently implemented using libvirt in the nova project) is not a secure virtualization technology for multi-tenant environments (specifically, containers may affect resource quotas for other containers hosted on the same machine). Additional containment technologies, such as AppArmor, may be used to provide better isolation between containers, although this is not the case by default. For all these reasons, the choice of this virtualization technology is not recommended in production.

If your compute hosts do not have hardware support for virtualization, LXC will likely provide better performance than QEMU. In addition, if your guests need to access to specialized hardware (e.g., GPUs), this may be easier to achieve with LXC than other hypervisors.



### Note

Some OpenStack Compute features may be missing when running with LXC as the hypervisor. See the [hypervisor support matrix](#) for details.

To enable LXC, ensure the following options are set in `/etc/nova/nova.conf` on all hosts running the `nova-compute` service.

```
compute_driver=libvirt.LibvirtDriver
libvirt_type=lxc
```

On Ubuntu 12.04, enable LXC support in OpenStack by installing the `nova-compute-lxc` package.

## VMware vSphere

### Introduction

OpenStack Compute supports the VMware vSphere product family. This section describes the additional configuration required to launch VMWare-based virtual machine images. vSphere versions 4.1 and greater are supported.

There are two OpenStack Compute drivers that can be used with vSphere:

- `vmwareapi.VMwareVCDriver`: a driver that lets `nova-compute` communicate with a VMware vCenter server managing a cluster of ESX hosts. With this driver and access to shared storage, advanced vSphere features like vMotion, High Availability, and Dynamic

Resource Scheduling (DRS) are available. With this driver, one nova-compute service is run per vCenter cluster.

- `vmwareapi.VMwareESXDriver`: a driver that lets nova-compute communicate directly to an ESX host, but does not support advanced VMware features. With this driver, one nova-compute service is run per ESX host.

## Prerequisites

You will need to install the following software installed on each nova-compute node:

- `python-suds`: This software is needed by the nova-compute service to communicate with vSphere APIs. If not installed, the "nova-compute" service shuts down with the message: "Unable to import suds".
- Tomcat server: This is required to serve up a local version of the vSphere WSDL file (see below).

On ubuntu, these packages can be installed by running:

```
sudo apt-get install python-suds tomcat6
```

Next, download the the SDK from <http://www.vmware.com/support/developer/vc-sdk/> and copy it into `/var/lib/tomcat6/webapps`. You should ensure that the WSDL is available, in eg `/var/lib/tomcat6/webapps/vmware/SDK/wsdl/vim25/vimService.wsdl`. Below we will point `nova.conf` to fetch this WSDL file from Tomcat using a URL pointing to localhost.

## Using the VMwareVCDriver

This section covers details of using the VMwareVCDriver.

### VMWareVCDriver configuration options

When using the VMwareVCDriver (i.e., vCenter) with OpenStack Compute, `nova.conf` must include the following VMWare-specific config options:

```
vmwareapi_host_ip=<vCenter host IP>
vmwareapi_host_username=< vCenter username>
vmwareapi_host_password=< vCenter password>
vmwareapi_cluster_name=< vCenter cluster name>
vmwareapi_wsdl_loc=http://127.0.0.1:8080/vmware/SDK/wsdl/vim25/vimService.wsdl
```

Remember that you will have only one nova-compute service per cluster. It is recommended that this host run as a VM with high-availability enabled as part of that same cluster.

Also note that many of the `nova.conf` options mentioned elsewhere in this document that are relevant to libvirt do not apply to using this driver.

## Requirements + Limitations

The VMwareVCDriver is new in Grizzly, and as a result there are some important deployment requirements and limitations to be aware of. In many cases, these items will be addressed in future releases.

- Each cluster can only be configured with a single Datastore. If multiple Datastores are configured, the first one returned via the vSphere API will be used.
- Because a single nova-compute is used per cluster, the nova-scheduler views this as a single host with resources amounting to the aggregate resources of all ESX hosts managed by the cluster. This may result in unexpected behavior depending on your choice of scheduler.
- Security Groups are not supported if Nova-Network is used. Security Groups are only supported if the VMware driver is used in conjunction with the the OpenStack Networking Service running the Nicira NVP plugin.

## Using the VMwareESXDriver

This section covers details of using the VMwareESXDriver.

### VMWareESXDriver configuration options

When using the VMwareESXDriver (i.e., no vCenter) with OpenStack Compute, configure nova.conf with the following VMWare-specific config options:

```
vmwareapi_host_ip=<ESXi host IP>
vmwareapi_host_username=< ESXi host username>
vmwareapi_host_password=< ESXi host password>
vmwareapi_wsdl_loc=http://127.0.0.1:8080/vmware/SDK/wsdl/vim25/vimService.wsdl
```

Remember that you will have one nova-compute service per ESXi host. It is recommended that this host run as a VM on the same ESXi host it is managing.

Also note that many of the nova.conf options mentioned elsewhere in this document that are relevant to libvirt do not apply to using this driver.

### Requirements + Limitations

The ESXDriver is unable to take advantage of many of the advanced capabilities associated with the vSphere platform, namely vMotion, High Availability, and Dynamic Resource Scheduler (DRS).

## Images with VMware vSphere

When using either VMware driver, images should be uploaded to the OpenStack Image Service in the VMDK format. For example:

```
glance add name="ubuntuLTS" disk_format=vmddk container_format=ovf \
is_public=true vmware_adaptertype="lsiLogic" vmware_disktype="preallocated" \
vmware_ostype="ubuntu64Guest" < ubuntuLTS-flat.vmdk
```

## Networking with VMware vSphere

The VMware driver support networking with both Nova-Network and the Openstack Networking Service.

- If using nova-network with the FlatManager or FlatDHCPManager, before provisioning VMs, create a port group with the same name as the 'flat\_network\_bridge' value in nova.conf (default is 'br100'). All VM NICs will be attached to this port group.
- If using nova-network with the VlanManager, before provisioning VMs, make sure the 'vmwareapi\_vlan\_interface' configuration option is set to match the ESX host interface that will handle VLAN-tagged VM traffic. OpenStack Compute will automatically create the corresponding port groups.
- If using the OpenStack Networking Service, before provisioning VMs, create a port group with the same name as the 'vmware.integration\_bridge' value in nova.conf (default is 'br-int'). All VM NICs will be attached to this port group for management by the OpenStack Networking Plugin.

## Volumes with VMware vSphere

The VMware driver has limited support for attaching Volumes from the OpenStack Block Storage service, supporting attachments only if the volume driver type is 'iscsi'. There is no support for volumes based on vCenter Datastores in this release.

## PowerVM

### Introduction

PowerVM compute driver connects to an Integrated Virtualization Manager (IVM) to perform PowerVM Logical Partition (LPAR) deployment and management. The driver supports file-based deployment using images from Glance.



#### Note

Hardware Management Console (HMC) is not yet supported.

For more detailed information about PowerVM Virtualization system, refer to the IBM Redbook publication: [IBM PowerVM Virtualization Introduction and Configuration](#).

### Configuration

To enable the PowerVM compute driver, add the following configuration options /etc/nova/nova.conf:

```
compute_driver=nova.virt.powervm.PowerVMDriver
powervm_mgr_type=ivm
powervm_mgr=powervm_hostname_or_ip_address
powervm_mgr_user=padmin
powervm_mgr_passwd=padmin_user_password
powervm_img_remote_path=/path/to/remote/image/directory
powervm_img_local_path=/path/to/local/image/directory/on/compute/host
```

## Hyper-V Virtualization Platform

It is possible to use Hyper-V as a compute node within an OpenStack Deployment. The nova-compute service runs as "openstack-compute," a 32bit service directly upon the

Windows platform with the Hyper-V role enabled. The necessary Python components as well as the nova-compute service are installed directly onto the Windows platform. Windows Clustering Services are not needed for functionality within the OpenStack infrastructure. The use of the Windows Server 2012 platform is recommended for the best experience and is the platform for active development. The following Windows platforms have been tested as compute nodes:

- **Windows Server 2008r2**

Both Server and Server Core with the Hyper-V role enabled (Shared Nothing Live migration is not supported using 2008r2)

- **Windows Server 2012**

Server and Core (with the Hyper-V role enabled), and Hyper-V Server

## Hyper-V Configuration

The following sections discuss how to prepare the Windows Hyper-V node for operation as an OpenStack Compute node. Unless stated otherwise, any configuration information should work for both the Windows 2008r2 and 2012 platforms.

### Local Storage Considerations

The Hyper-V compute node needs to have ample storage for storing the virtual machine images running on the compute nodes. You may use a single volume for all, or partition it into an OS volume and VM volume. It is up to the individual deploying to decide.

## Configure NTP

Network time services must be configured to ensure proper operation of the Hyper-V compute node. To set network time on your Hyper-V host you will need to run the following commands

```
C:\net stop w32time
```

```
C:\w32tm /config /manualpeerlist:pool.ntp.org,0x8 /syncfromflags:MANUAL
```

```
C:\net start w32time
```

## Configuring Hyper-V Virtual Switching

Information regarding the Hyper-V virtual Switch can be located here: <http://technet.microsoft.com/en-us/library/hh831823.aspx>

To quickly enable an interface to be used as a Virtual Interface the following PowerShell may be used:

```
PS C:\$if = Get-NetIPAddress -IPAddress 192* | Get-NetIPInterface
```

```
PS C:\New-VMSwitch -NetAdapterName $if.ifAlias -Name yourbridgename -  
AllowManagementOS $false
```

## Enable iSCSI Initiator Service

To prepare the Hyper-V node to be able to attach to volumes provided by cinder you must first make sure the Windows iSCSI initiator service is running and started automatically.

```
C:\sc start MSiSCSI
```

```
C:\sc config MSiSCSI start="auto"
```

## Configuring Shared Nothing Live Migration

Detailed information on the configuration of live migration can be found here: <http://technet.microsoft.com/en-us/library/jj134199.aspx>

The following outlines the steps of shared nothing live migration.

1. The target hosts ensures that live migration is enabled and properly configured in Hyper-V.
2. The target hosts checks if the image to be migrated requires a base VHD and pulls it from Glance if not already available on the target host.
3. The source hosts ensures that live migration is enabled and properly configured in Hyper-V.
4. The source hosts initiates a Hyper-V live migration.
5. The source hosts communicates to the manager the outcome of the operation.

The following two configuration options/flags are needed in order to support Hyper-V live migration and must be added to your nova.conf on the Hyper-V compute node:

- instances\_shared\_storage=False

This needed to support "shared nothing" Hyper-V live migrations. It is used in nova/compute/manager.py

- limit\_cpu\_features=True

This flag is needed to support live migration to hosts with different CPU features. This flag is checked during instance creation in order to limit the CPU features used by the VM.

- instances\_path=DRIVELETTER:\PATH\TO\YOUR\INSTANCES

### Additional Requirements:

- Hyper-V 2012 RC or Windows Server 2012 RC with Hyper-V role enabled
- A Windows domain controller with the Hyper-V compute nodes as domain members
- The instances\_path command line option/flag needs to be the same on all hosts
- The openstack-compute service deployed with the setup must run with domain credentials. You can set the service credentials with:

```
C:\sc config openstack-compute obj="DOMAIN\username" password="password"
```

### How to setup live migration on Hyper-V

To enable shared nothing live migration run the 3 PowerShell instructions below on each Hyper-V host:

```
PS C:\Enable-VMMigration
```

```
PS C:\Set-VMMigrationNetwork IP_ADDRESS
```

```
PS C:\Set-VMHost -VirtualMachineMigrationAuthenticationTypeKerberos
```



### Note

Please replace the IP\_ADDRESS with the address of the interface which will provide the virtual switching for nova-network.

### Additional Reading

Here's an article that clarifies the various live migration options in Hyper-V:

<http://ariessysadmin.blogspot.ro/2012/04/hyper-v-live-migration-of-windows.html>

## "Python Requirements">

### Python

Python 2.7.3 must be installed prior to installing the OpenStack Compute Driver on the Hyper-V server. Download and then install the MSI for windows here:

- <http://www.python.org/ftp/python/2.7.3/python-2.7.3.msi>
- Install the MSI accepting the default options.
- The installation will put python in C:/python27.

### Setup tools

You will require pip to install the necessary python module dependencies. The installer will install under the C:\python27 directory structure. Setuptools for Python 2.7 for Windows can be download from here:

<http://pypi.python.org/packages/2.7/s/setuptools/setuptools-0.6c11.win32-py2.7.exe>

### Python Dependencies

The following packages need to be downloaded and manually installed onto the Compute Node

- **MySQL-python**

<http://codegood.com/download/10/>

- **pywin32**

Download and run the installer from the following location

<http://sourceforge.net/projects/pywin32/files/pywin32/Build%20217/pywin32-217.win32-py2.7.exe>

- **greenlet**

Select the link below:

<http://www.lfd.uci.edu/~gohlke/pythonlibs/>

You will need to scroll down to the greenlet section for the following file:  
greenlet-0.4.0.win32-py2.7.#exe

Click on the file, to initiate the download. Once the download is complete, run the installer.

The following python packages need to be installed via easy\_install or pip. Run the following replacing PACKAGE\_NAME with the packages below:

```
C:\c:\Python27\Scripts\pip.exe install PACKAGE_NAME
```

- amqplib
- anyjson
- distribute
- eventlet
- httplib2
- iso8601
- jsonschema
- kombu

- netaddr
- paste
- paste-deploy
- prettytable
- python-cinderclient
- python-glanceclient
- python-keystoneclient
- repoze.lru
- routes
- sqlalchemy
- simplejson
- warlock
- webob
- wmi

## Installing Nova-compute

### Using git on Windows to retrieve source

Git be used to download the necessary source code. The installer to run Git on Windows can be downloaded here:

<http://code.google.com/p/msysgit/downloads/list?q=full+installer+official+git>

Download the latest installer. Once the download is complete double click the installer and follow the prompts in the installation wizard. The default should be acceptable for the needs of the document.

Once installed you may run the following to clone the Nova code.

```
C:\git.exe clone https://github.com/openstack/nova.git
```

## Configuring Nova.conf

The `nova.conf` file must be placed in `C:\etc\nova` for running OpenStack on Hyper-V. Below is a sample `nova.conf` for Windows:

```
[DEFAULT]
```

```
verbose=true
force_raw_images=false
auth_strategy=keystone
fake_network=true
vswitch_name=openstack-br
logdir=c:\openstack\
state_path=c:\openstack\
lock_path=c:\openstack\
instances_path=e:\Hyper-V\instances
policy_file=C:\Program Files (x86)\OpenStack\nova\etc\nova\policy.json
api_paste_config=c:\openstack\nova\etc\nova\api-paste.ini
rabbit_host=IP_ADDRESS
glance_api_servers=IP_ADDRESS:9292
image_service=nova.image.glance.GlanceImageService
sql_connection=mysql://nova:passwd@IP_ADDRESS/nova
instances_shared_storage=false
limit_cpu_features=true
compute_driver=nova.virt.hyperv.driver.HyperVDriver
volume_api_class=nova.volume.cinder.API
```

## Preparing Images for use with Hyper-V

Hyper-V currently supports only the VHD file format for virtual machine instances. Detailed instructions for installing virtual machines on Hyper-V can be found here:

<http://technet.microsoft.com/en-us/library/cc772480.aspx>

Once you have successfully created a virtual machine, you can then upload the image to glance using the native glance-client:

```
C:\glance image-create --name="VM_IMAGE_NAME" --is-public=true --container-format=bare --disk-format=vhd
```

## Running Compute with Hyper-V

To start the nova-compute service, run this command from a console in the Windows server:

```
C:\C:\python27\python.exe c:\openstack\nova\bin\nova-compute.py
```

## Troubleshooting Hyper-V Configuration

- I ran the nova-manage service list command from my controller; however, I'm not seeing smiley faces for Hyper-V compute nodes, what do I do?

*Verify that you are synchronized with a network time source. Instructions for configuring NTP on your Hyper-V compute node are located [here](#)*

## Bare Metal Driver

The baremetal driver is a hypervisor driver for Openstack Nova Compute. Within the Openstack framework, it has the same role as the drivers for other hypervisors (libvirt, xen, etc), and yet it is presently unique in that the hardware is not virtualized - there is no hypervisor between the tenants and the physical hardware. It exposes hardware via Openstack's API, using pluggable sub-drivers to deliver machine imaging (PXE) and power control (IPMI). With this, provisioning and management of physical hardware is accomplished using common cloud APIs and tools, such as Heat or salt-cloud. However, due to this unique situation, using the baremetal driver requires some additional preparation of its environment, the details of which are beyond the scope of this guide.



### Note

Some OpenStack Compute features are not implemented by the baremetal hypervisor driver. See the [hypervisor support matrix](#) for details.

For the Baremetal driver to be loaded and function properly, ensure that the following options are set in `/etc/nova/nova.conf` on your `nova-compute` hosts.

```
[default]
compute_driver=nova.virt.baremetal.driver.BareMetalDriver
firewall_driver = nova.virt.firewall.NoopFirewallDriver
scheduler_host_manager=nova.scheduler.baremetal_host_manager.
BaremetalHostManager
    ram_allocation_ratio=1.0
    reserved_host_memory_mb=0
```

There are many configuration options specific to the Baremetal driver. Also, some additional steps will be required, such as building the baremetal deploy ramdisk. See the [main wiki page](#) for details and implementation suggestions.

## Nova Compute Fibre Channel Support

### Overview of Fibre Channel Support

- Fibre Channel support in OpenStack Compute is remote block storage attached to Compute nodes for VMs.
- Fibre Channel supports the KVM hypervisor in only the grizzly release.
- There is no automatic zoning support in Nova or Cinder for Fibre Channel. Fibre Channel arrays must be pre-zoned or directly attached to the KVM hosts.

### Requirements for KVM Hosts

The KVM host must have the following system packages installed:

- `sysfstoools` - Nova uses the `systool` application in this package.
- `sg3-utils` - Nova uses the `sg_scan` and `sginfo` applications.

Installing the `multipath-tools` package is optional.

## Installing the Required Packages

Use the following commands to install the system packages.

- For systems running Ubuntu:

```
$sudo apt-get install sysfstools sg3-utils multipath-tools
```

- For systems running Red Hat:

```
$sudo yum install sysfstools sg3_utils multipath-tools
```

# 10. Networking with nova-network

## Table of Contents

Networking Options .....	201
DHCP server: dnsmasq .....	204
Metadata service .....	205
Configuring Networking on the Compute Node .....	208
Enabling Ping and SSH on VMs .....	235
Configuring Public (Floating) IP Addresses .....	236
Removing a Network from a Project .....	238
Using multiple interfaces for your instances (multinic) .....	238
Existing High Availability Options for Networking .....	242
Troubleshooting Networking .....	245

By understanding the available networking configuration options you can design the best configuration for your OpenStack Compute instances.

## Networking Options

This section offers a brief overview of each concept in networking for Compute. With the Grizzly release, you can chose either to install and configure nova-network for networking between VMs or use the Networking service (quantum) for networking. Refer to the [Network Administration Guide](#) to configure Compute networking options with Quantum.

For each VM instance, Compute assigns to it a private IP address. (Currently, Compute with nova-network only supports Linux bridge networking that allows the 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.

Currently, Compute with nova-network supports three kinds of networks, implemented in three “Network Manager” types:

- Flat Network Manager
- Flat DHCP Network Manager
- VLAN Network Manager

The three kinds of networks can co-exist in a cloud system. However, since you can't yet select the type of network for a given project, you cannot configure more than one type of network in a given Compute installation.



### Note

All of the networking options require network connectivity to be already set up between OpenStack physical nodes. OpenStack will not configure any physical

network interfaces. OpenStack will automatically create all network bridges (i.e., br100) and VM virtual interfaces.

All machines must have a *public* and *internal* network interface (controlled by the options: `public_interface` for the public interface, and `flat_interface` and `vlan_interface` for the internal interface with flat / VLAN managers).

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

All the network managers configure the network using *network drivers*, e.g. the linux L3 driver (`l3.py` and `linux_net.py`) which makes use of `iptables`, `route` and other network management facilities, and also of libvirt's [network filtering facilities](#). The driver isn't tied to any particular network manager; all network managers use the same driver. The driver usually initializes (creates bridges etc.) 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, there is just 1 instance of `nova-network` which is used as a default gateway for VMs and hosts a single DHCP server (`dnsmasq`), whereas in multi-host mode every compute node has its own `nova-network`. In any case, all traffic between VMs and the outer world flows through `nova-network`. There are pros and cons to both modes, read more in [Existing High Availability Options](#).

Compute makes a distinction between *fixed IPs* and *floating IPs* for VM instances. Fixed IPs are IP addresses that are assigned to an instance on creation and stay the same until the instance is explicitly terminated. By contrast, floating IPs are addresses that can be dynamically associated with an instance. A floating IP address can be disassociated and associated with another instance at any time. A user can reserve a floating IP for their project.

In **Flat Mode**, a network administrator specifies a subnet. The IP addresses for VM instances are grabbed 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 may create the Linux networking bridge (typically named `br100`, although this configurable) on the systems running the `nova-network` service. All instances of the system are attached to the same bridge, configured manually by the network administrator.



## Note

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

In **Flat DHCP Mode**, OpenStack starts a DHCP server (`dnsmasq`) to pass out IP addresses to VM instances from the specified subnet in addition to manually configuring the networking bridge. IP addresses for VM instances are grabbed from a subnet specified by the network administrator.

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

In any setup with flat networking, the host(-s) with nova-network on it is (are) responsible for forwarding traffic from the private network. Compute can determine the NAT entries for each network when you have `fixed_range=''` in your `nova.conf`. Sometimes NAT is not used, such as when `fixed_range` is configured with all public IPs and a hardware router is used (one of the HA options). Such host(-s) needs to have `br100` configured and physically connected to any other nodes that are hosting VMs. You must set the `flat_network_bridge` option or create networks with the `bridge` parameter in order to avoid raising an error. Compute nodes have iptables/ebtables entries created per project and instance to protect against IP/MAC address spoofing and ARP poisoning.



## Note

To use the new dynamic `fixed_range` setup in Grizzly, set `fixed_range=''` in your `nova.conf`. For backwards compatibility, Grizzly supports the `fixed_range` option and if set will perform the default logic from Folsom and earlier releases.



## Note

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

**VLAN Network Mode is the default mode** for OpenStack Compute. In this mode, Compute creates a VLAN and bridge for each project. For multiple machine installation, the VLAN Network Mode requires a switch that supports VLAN tagging (IEEE 802.1Q). The project 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 project, 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 project's instances that can be accessed via a dedicated VPN connection from the Internet. In this mode, each project gets its own VLAN, Linux networking bridge, and subnet.

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



## Note

With the default Compute settings, once a virtual machine instance is destroyed, it can take some time for the IP address associated with the destroyed instance to become available for assignment to a new instance.

The `force_dhcp_release=True` configuration option, when set, causes the Compute service to send out a DHCP release packet when it destroys a virtual machine instance. The result is that the IP address assigned to the instance is immediately released.

This configuration option applies to both Flat DHCP mode and VLAN Manager mode.

Use of this option requires the `dhcp_release` program. Verify that this program is installed on all hosts running the `nova-compute` service before enabling this option. This can be checked with the `which` command, and will return the complete path if the program is installed. As root:

```
# which dhcp_release
/usr/bin/dhcp_release
```

## DHCP server: dnsmasq

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

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

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

See the [high availability section](#) for an example of how to change the behavior of dnsmasq using a dnsmasq configuration file. The dnsmasq documentation has a more comprehensive [dnsmasq configuration file example](#).

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

```
dns_server=8.8.8.8
```

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

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

# Metadata service

## Introduction

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

To retrieve a list of supported versions for the OpenStack metadata API, make a GET request to

```
http://169.254.169.254/openstack
```

For example:

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

To retrieve a list of supported versions for the EC2-compatible metadata API, make a GET request to

```
http://169.254.169.254
```

For example:

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

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

## OpenStack metadata API

Metadata from the OpenStack API is distributed in JSON format. To retrieve the metadata, make a GET request to

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

For example:

```
$ curl http://169.254.169.254/openstack/2012-08-10/meta_data.json
{"uuid": "d8e02d56-2648-49a3-bf97-6be8f1204f38", "availability_zone": "nova", "hostname": "test.novalocal", "launch_index": 0, "meta": {"priority": "low", "role": "webserver"}, "public_keys": {"mykey": "ssh-rsa AAAAB3NzaC1yc2EAAAQABAAQgQDYVEprvtYJXVOBN0XNKVVRNCRX6BlnNbI"}}
```

```
+USLGaislsUWPwtSg7z9K9vhbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq8Oh5b2i71/
3BISpyxTBH/uZDHs1W2a+SrPDCeumMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated
by Nova\n" }, "name": "test" }
```

Here is the same content after having run through a JSON pretty-printer:

```
{
    "availability_zone": "nova",
    "hostname": "test.novalocal",
    "launch_index": 0,
    "meta": {
        "priority": "low",
        "role": "webserver"
    },
    "name": "test",
    "public_keys": {
        "mykey": "ssh-rsa
AAAAB3NzaC1yc2EAAAQABAAAAgQDYVEprvtYJXVOBN0XNKVVRNCRX6BlnNbI
+USLGaislsUWPwtSg7z9K9vhbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq8Oh5b2i71/
3BISpyxTBH/uZDHs1W2a+SrPDCeumMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated
by Nova\n"
    },
    "uuid": "d8e02d56-2648-49a3-bf97-6be8f1204f38"
}
```

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

```
http://169.254.169.254/openstack/2012-08-10/user_data
```

For example:

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

## EC2 metadata API

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

The EC2 API exposes a separate URL for each metadata. A listing of these elements can be retrieved by making a GET query to:

```
http://169.254.169.254/2009-04-04/meta-data/
```

For example:

```
$ curl http://169.254.169.254/2009-04-04/meta-data/
ami-id
ami-launch-index
ami-manifest-path
block-device-mapping/
hostname
instance-action
instance-id
instance-type
kernel-id
```

```
local-hostname
local-ipv4
placement/
public-hostname
public-ipv4
public-keys/
ramdisk-id
reservation-id
security-groups
$ curl http://169.254.169.254/2009-04-04/meta-data/block-device-mapping/
ami
$ curl http://169.254.169.254/2009-04-04/meta-data/placement/
availability-zone
$ curl http://169.254.169.254/2009-04-04/meta-data/public-keys/
0=mykey
```

Instances can retrieve the public SSH key (identified by keypair name when a user requests a new instance) by making a GET request to:

```
http://169.254.169.254/2009-04-04/meta-data/public-keys/0/openssh-key
```

For example:

```
$ curl http://169.254.169.254/2009-04-04/meta-data/public-keys/0/openssh-key
ssh-rsa AAAAB3NzaC1yc2EAAAQABAAAAgQDYVEprvtYJXVOBN0XNKVRNCRX6B1nNbI
+USLGaislsUWPwtSg7z9K9vhbYAPUZcq8c/s5S9dg5vTHbsiyPCIDOKyeHba4MUJq8Oh5b2i71/
3BISPyxTBH/uZDHds1W2a+SrPDCeumMoss9NFhBdKtDkdG9zyi0ibmCP6yMdEX8Q== Generated
by Nova
```

Instances can retrieve user data by making a GET request to:

```
http://169.254.169.254/2009-04-04/user-data
```

For example:

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

## Running the metadata service

The metadata service is implemented by either the `nova-api` service or the `nova-api-metadata` service. (The `nova-api-metadata` service is generally only used when running in multi-host mode, see the section titled [Existing High Availability Options for Networking](#) for details). If you are running the `nova-api` service, you must have `metadata` as one of the elements of the list of the `enabled_apis` configuration option in `/etc/nova/nova.conf`. The default `enabled_apis` configuration setting includes the `metadata` service, so you should not need to modify it.

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



### Warning

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



### Note

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

Set the metadata\_host configuration option to the IP address of the host where the nova-api service is running.

## Configuring Networking on the Compute Node

To configure the Compute node's networking for the VM images, the overall steps are:

1. Set the network\_manager option in nova.conf.
2. Use the nova network-create label --fixed-range-v4 CIDR [--vlan *vlan\_id*] command to create the subnet that the VMs reside on, specifying a VLAN if running in VLAN Network Mode.
3. Integrate the bridge with your network.

By default, Compute uses the VLAN Network Mode. You choose the networking mode for your virtual instances in the nova.conf file. Here are the three possible options:

- --network\_manager=nova.network.manager.FlatManager  
Simple, non-VLAN networking
- --network\_manager=nova.network.manager.FlatDHCPManager  
Flat networking with DHCP, you must set a bridge using the flat\_network\_bridge option
- --network\_manager=nova.network.manager.VlanManager  
VLAN networking with DHCP. This is the Default if no network manager is defined in nova.conf.

Use the following command to create a subnet (named *private* in this example) that your VMs will run on :

```
nova network-create private --fixed-range-v4 192.168.0.0/24
```

When using the XenAPI compute driver, the OpenStack services run in a virtual machine. This means networking is significantly different when compared to the networking with the libvirt compute driver. Before reading how to configure networking using the XenAPI compute driver, you may find it useful to read the Citrix article on [Understanding XenServer Networking](#) and the section of this document that describes [XenAPI and OpenStack](#).

## Configuring Flat Networking

FlatNetworking uses ethernet adapters configured as bridges to allow network traffic to transit between all the various nodes. This setup can be done with a single adapter

on the physical host, or multiple. This option does not require a switch that does VLAN tagging as VLAN networking does, and is a common development installation or proof of concept setup. When you choose Flat networking, Nova does not manage networking at all. Instead, IP addresses are injected into the instance via the file system (or passed in via a guest agent). Metadata forwarding must be configured manually on the gateway if it is required within your network.

To configure flat networking, ensure that your `nova.conf` file contains the following line:

```
network_manager=nova.network.manager.FlatManager
```



### Note

When configuring Flat Networking, failing to enable `flat_injected` can prevent guest VMs from receiving their IP information at boot time.

## Libvirt Flat Networking

Compute defaults to a bridge device named 'br100' which is stored in the Nova database, so you can change the name of the bridge device by modifying the entry in the database. Consult the diagrams for additional configuration options.

In any set up with FlatNetworking (either Flat or FlatDHCP), the host with `nova-network` on it is responsible for forwarding traffic from the private network. Compute determines the "fixed range" for IPs configured dynamically by pulling from the configured networks when you set `fixed_range=' '` in `nova.conf`. This dynamic range configuration allows for non-contiguous subnets to be configured in the `fixed_ip` space and only configures the NAT rules as they are needed. This also restricts the NAT range to the smallest range required preventing the NAT from impacting subnets that might exist on the external network.

This host needs to have `br100` configured and talking to any other nodes that are hosting VMs. With either of the Flat Networking options, the default gateway for the virtual machines is set to the host which is running `nova-network`.

Set the compute node's external IP address to be on the bridge and add `eth0` to that bridge. To do this, edit your network interfaces configuration to look like the following example:

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

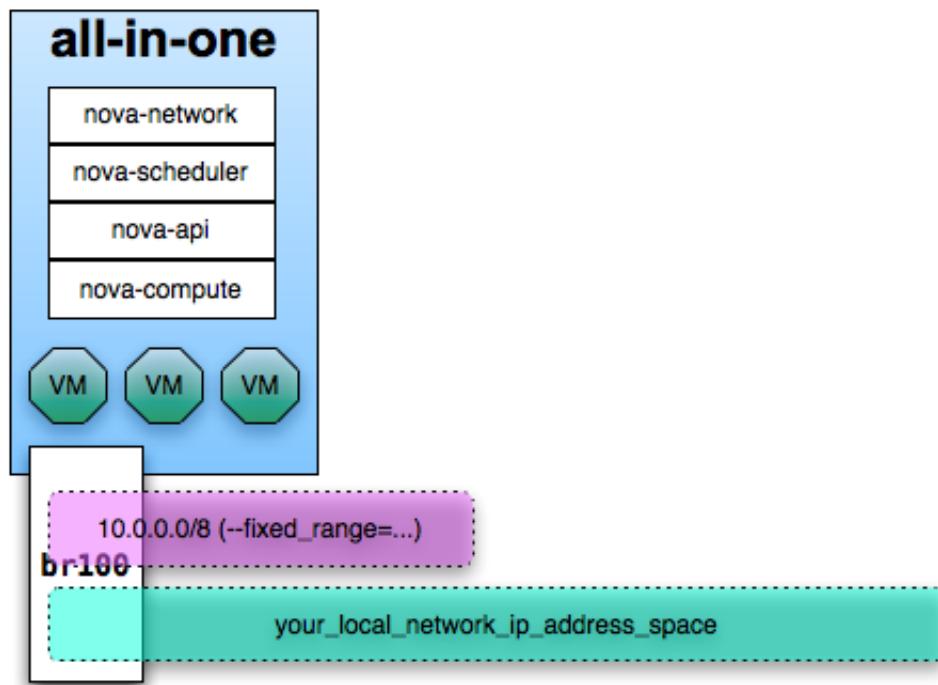
# Networking for OpenStack Compute
auto br100

iface br100 inet dhcp
    bridge_ports      eth0
    bridge_stp        off
    bridge_maxwait   0
    bridge_fd         0
```

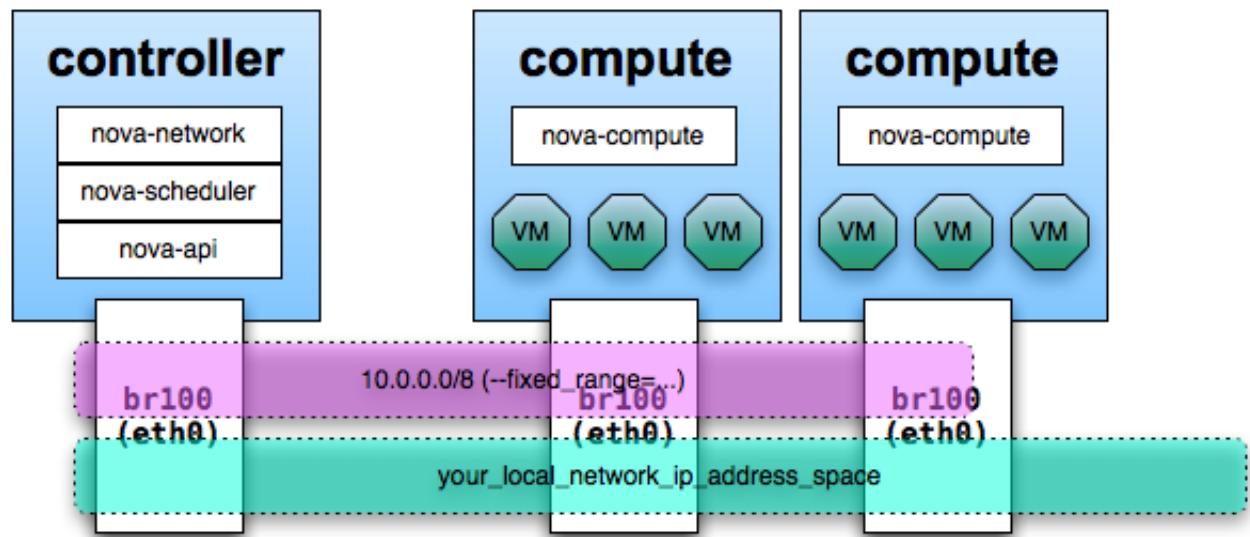
Next, restart networking to apply the changes: `sudo /etc/init.d/networking restart`

For an all-in-one development setup, this diagram represents the network setup.

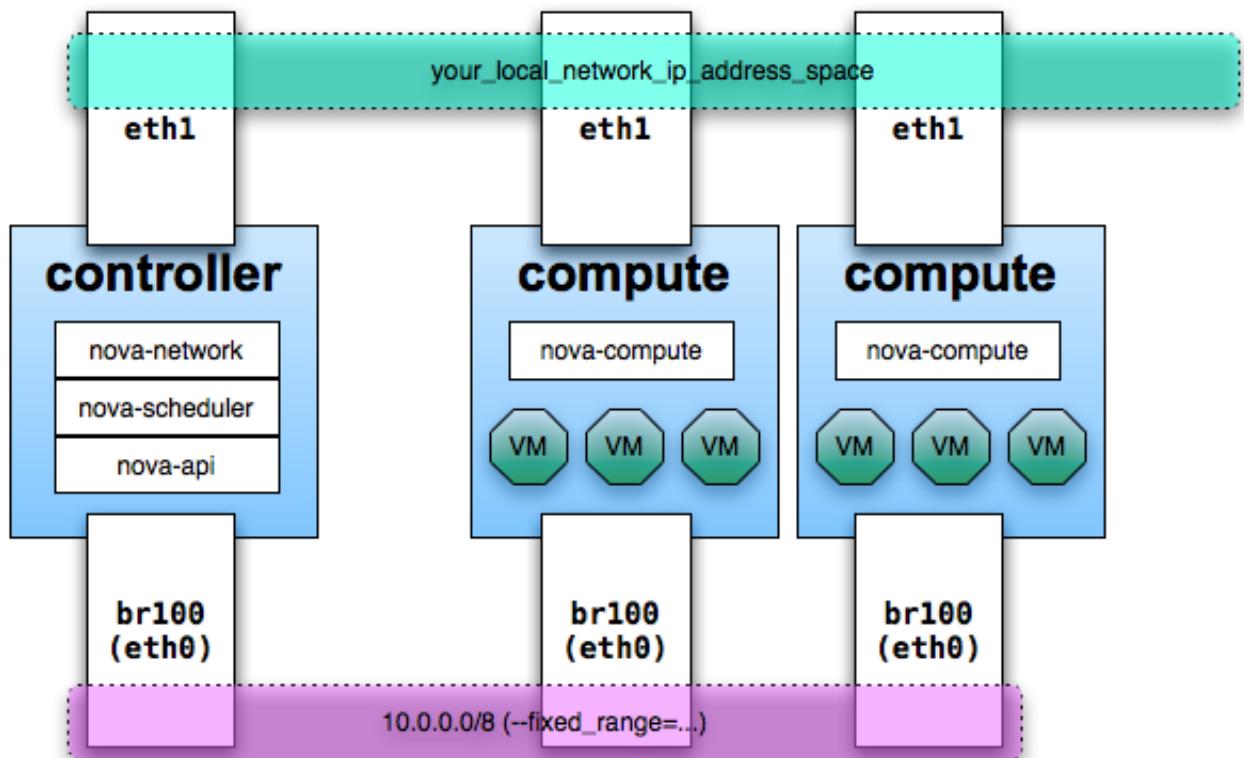
**Figure 10.1. Flat network, all-in-one server installation**



For multiple compute nodes with a single network adapter, which you can use for smoke testing or a proof of concept, this diagram represents the network setup.

**Figure 10.2. Flat network, single interface, multiple servers**

For multiple compute nodes with multiple network adapters, this diagram represents the network setup. You may want to use this setup for separate admin and data traffic.

**Figure 10.3. Flat network, multiple interfaces, multiple servers**

## XenAPI Flat Networking

When using the XenAPI driver, the virtual machines created by OpenStack are attached to the XenServer bridge configured in the `flat_network_bridge` setting. Otherwise, flat networking works in a very similar way with both the libvirt driver and the XenAPI driver.

## Configuring Flat DHCP Networking

With Flat DHCP, the host(-s) running nova-network act as the gateway to the virtual nodes. If you're using single-host networking, you can optionally set `network_host` on the `nova.conf` stored on the nova-compute node to tell it which host the nova-network is running on so it can more efficiently communicate with nova-network. In any setup with flat networking, the hosts with nova-network on it are responsible for forwarding traffic from the private network configured with the `fixed_range`= directive in `nova.conf` and the `flat_network_bridge` flag which you must also set to the name of the bridge (as there is no default). The nova-network service will track leases and releases in the database, using dnsmasq's `dhcp-script` facility (the script `bin/nova-dhcpbridge` is supplied) so it knows if a VM instance has stopped properly configuring via DHCP (e.g. when a DHCP lease expires, the fixed IP is released from the nova database). Lastly, it sets up iptables rules to allow the VMs to communicate with the outside world and contact a special metadata server to retrieve information from the cloud.

Compute hosts in the FlatDHCP model are responsible for bringing up a matching bridge and bridging the VM tap devices into the same ethernet device that the network host is on. The compute hosts should not have an IP address on the VM network, because the bridging puts the VMs and the network host on the same logical network. When a VM boots, the VM sends out DHCP packets, and the DHCP server on the network host responds with their assigned IP address (remember, the address is actually assigned by nova and put into DHCP server's configuration file, the DHCP server merely tells the VM what it is).

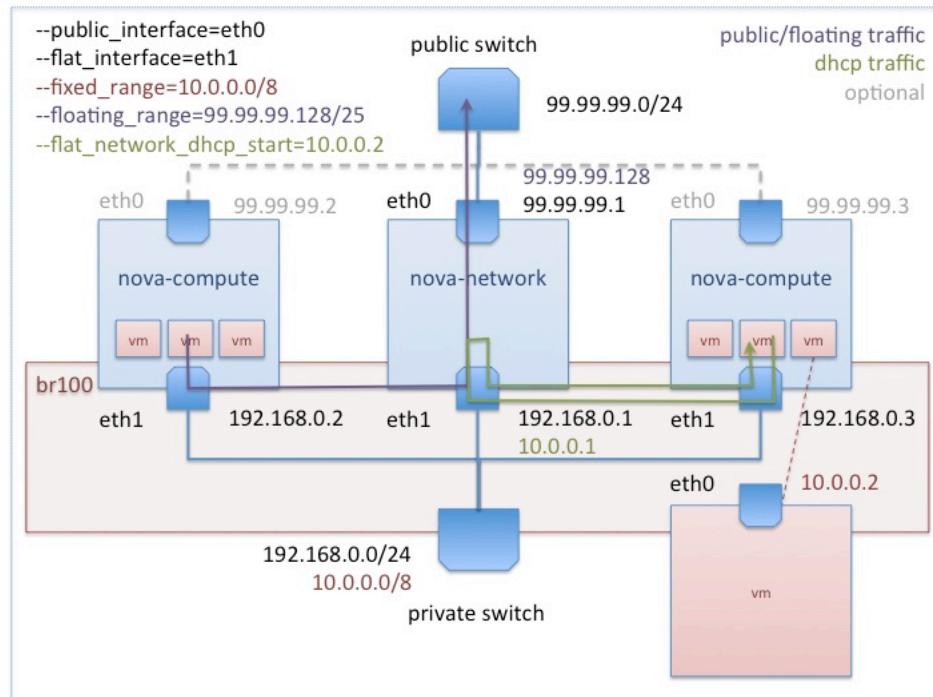
You can read a detailed walk-through of what exactly happens in single-host Flat DHCP mode in [this blogpost](#), parts of which are also relevant in other networking modes.

FlatDHCP doesn't create VLANs, it creates a bridge. This bridge works just fine on a single host, but when there are multiple hosts, traffic needs a way to get out of the bridge onto a physical interface.

## Libvirt Flat DHCP Networking

When using the libvirt driver, the setup will look like the figure below:

**Figure 10.4. Flat DHCP network, multiple interfaces, multiple servers with libvirt driver**



Be careful when setting up `--flat_interface`. If you specify an interface that already has an IP it will break and if this is the interface you are connecting through with SSH, you cannot fix it unless you have ipmi/console access. In FlatDHCP mode, the setting for `--network_size` should be number of IPs in the entire fixed range. If you are doing a /12 in CIDR notation, then this number would be  $2^{20}$  or 1,048,576 IP addresses. That said, it will take a very long time for you to create your initial network, as an entry for each IP will be created in the database.

If you have an unused interface on your hosts (eg eth2) that has connectivity with no IP address, you can simply tell FlatDHCP to bridge into the interface by specifying `flat_interface=<interface>` in your configuration file. The network host will automatically add the gateway ip to this bridge. If this is the case for you, edit your `nova.conf` file to contain the following lines:

```
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
network_manager=nova.network.manager.FlatDHCPManager
fixed_range=''
flat_network_bridge=br100
flat_interface=eth2
flat_injected=False
public_interface=eth0
```

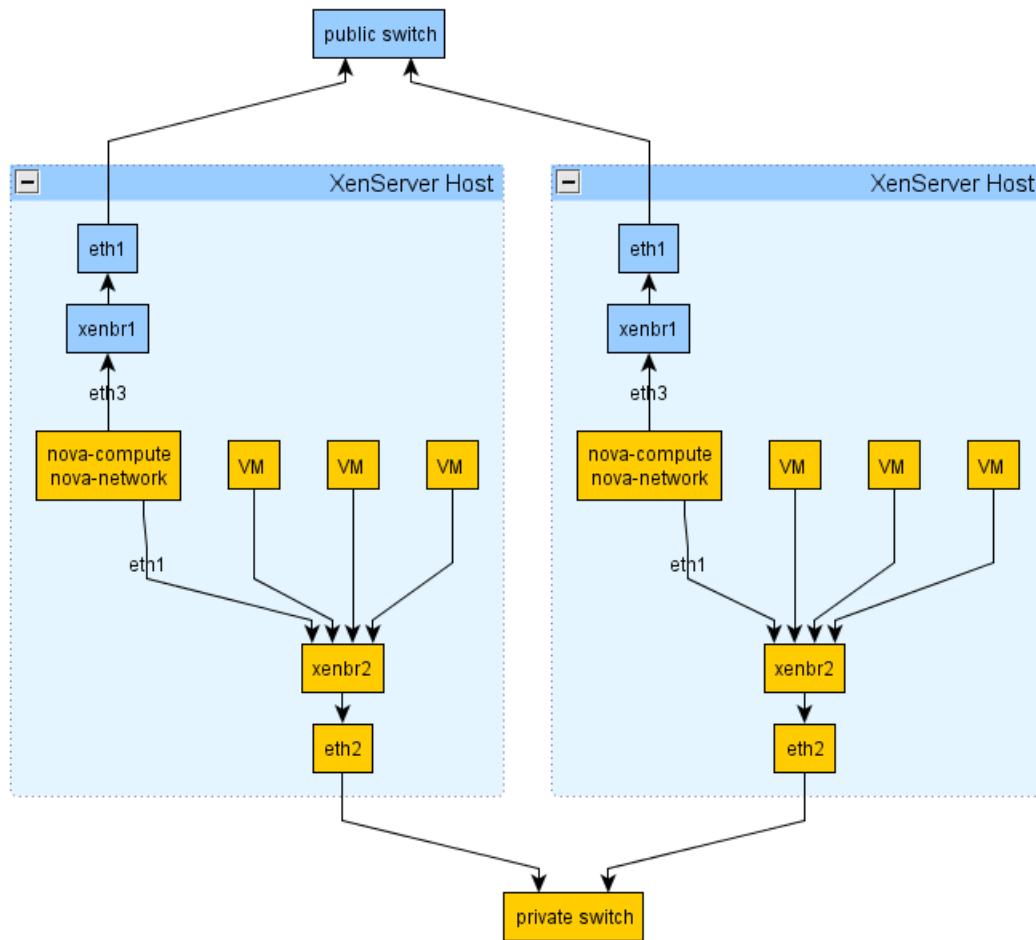
You can also add the unused interface to br100 manually and not set `flat_interface`.

Integrate your network interfaces to match this configuration.

## XenAPI Flat DHCP Networking

The following figure shows a setup with Flat DHCP networking, network HA, and using multiple interfaces. For simplicity, the management network (on XenServer eth0 and eth2 of the VM running the OpenStack services) has been omitted from the figure below.

**Figure 10.5. Flat DHCP network, multiple interfaces, multiple servers, network HA with XenAPI driver**



Here is an extract from a `nova.conf` file in a system running the above setup:

```
network_manager=nova.network.manager.FlatDHCPManager
xenapi_vif_driver=nova.virt.xenapi.vif.(XenAPIBridgeDriver or
    XenAPIOpenVswitchDriver)
flat_interface=eth1
flat_network_bridge=xenbr2
public_interface=eth3
multi_host=True
dhcpbridge_flagfile=/etc/nova/nova.conf
fixed_range=''
force_dhcp_release=True
send_arp_for_ha=True
flat_injected=False
firewall_driver=nova.virt.xenapi.firewall.Dom0IptablesFirewallDriver
```

You should notice that `flat_interface` and `public_interface` refer to the network interface on the VM running the OpenStack services, not the network interface on the Hypervisor.

Secondly `flat_network_bridge` refers to the name of XenAPI network that you wish to have your instance traffic on, i.e. the network on which the VMs will be attached. You can either specify the bridge name, such as `xenbr2`, or the name label, such as `vmbr`. Specifying the name-label is very useful in cases where your networks are not uniform across your XenServer hosts.

When you have a limited number of network cards on your server, it is possible to use networks isolated using VLANs for the public and network traffic. For example, if you have two XenServer networks `xapi1` and `xapi2` attached on VLAN 102 and 103 on `eth0`, respectively, you could use these for `eth1` and `eth3` on your VM, and pass the appropriate one to `flat_network_bridge`.

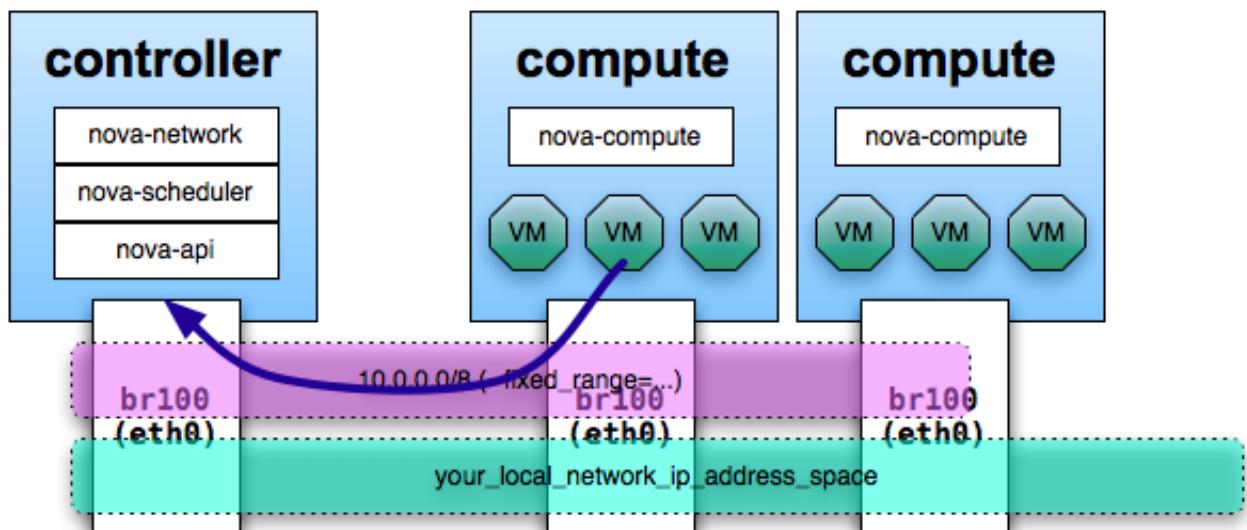
When using XenServer, it is best to use the firewall driver written specifically for XenServer. This pushes the firewall rules down to the hypervisor, rather than running them in the VM that is running `nova-network`.

## Outbound Traffic Flow with Any Flat Networking

In any set up with FlatNetworking, the host with `nova-network` on it is responsible for forwarding traffic from the private network dynamically determined by Compute with the `fixed_range=''` directive in `nova.conf`. This host needs to have a bridge interface (e.g., `br100`) configured and talking to any other nodes that are hosting VMs. With either of the Flat Networking options, the default gateway for the virtual machines is set to the host which is running `nova-network`.

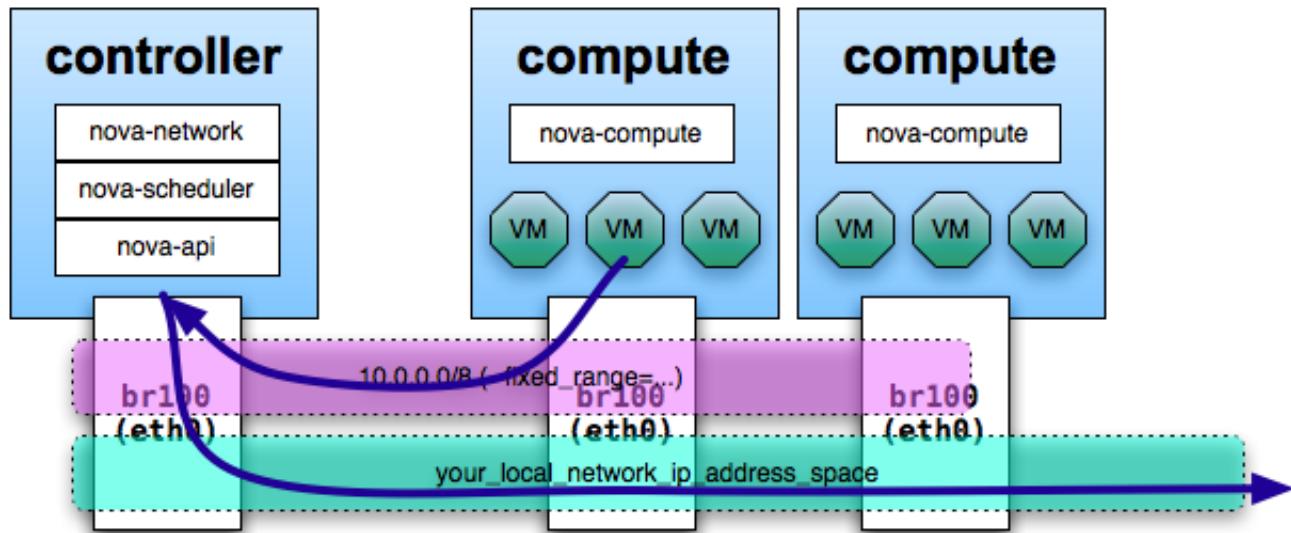
When a virtual machine sends traffic out to the public networks, it sends it first to its default gateway, which is where `nova-network` is configured.

**Figure 10.6. Single adaptor hosts, first route**



Next, the host on which nova-network is configured acts as a router and forwards the traffic out to the Internet.

**Figure 10.7. Single adaptor hosts, second route**



### Warning



If you're using a single interface, then that interface (often eth0) needs to be set into promiscuous mode for the forwarding to happen correctly. This does not appear to be needed if you're running with physical hosts that have and use two interfaces.

## Configuring VLAN Networking

Compute can be configured so that the virtual machine instances of different projects (tenants) are in different subnets, with each subnet having a different VLAN tag. This can be useful in networking environments where you have a large IP space which is cut up into smaller subnets. The smaller subnets are then trunked together at the switch level (dividing layer 3 by layer 2) so that all machines in the larger IP space can communicate. The purpose of this is generally to control the size of broadcast domains. It can also be useful to provide an additional layer of isolation in a multi-tenant environment.

### Note



The terms *network* and *subnet* are often used interchangeably in discussions of VLAN mode. In all cases, we are referring to a range of IP addresses specified by a *subnet* (e.g., 172.16.20.0/24) that are on the same VLAN (layer 2 network).

Running in VLAN mode is more complex than the other network modes. In particular:

- IP forwarding must be enabled

- The hosts running nova-network and nova-compute must have the 8021q kernel module loaded
- Your networking switches must support VLAN tagging
- Your networking switches must be configured to enable the specific VLAN tags you specify in your Compute setup
- You will need information about your networking setup from your network administrator to configure Compute properly (e.g., netmask, broadcast, gateway, ethernet device, VLAN IDs)

The `network_manager=nova.network.manager.VlanManager` option specifies VLAN mode, which happens to be the default networking mode.

The bridges that are created by the network manager will be attached to the interface specified by `vlan_interface`, the example above uses the `eth0` interface, which is the default.

The `fixed_range` option deprecated in Grizzly and should be set to `fixed_range=''` so that Nova determines a CIDR block which describes the IP address space for all of the instances: this space will be divided up into subnets. This range is typically a [private network](#). The example above uses the private range `172.16.0.0/12`.

The `network_size` option refers to the default number of IP addresses in each network, although this can be overridden at network creation time. The example above uses a network size of 256, which corresponds to a /24 network.

Networks are created with the `nova network-create` command. Here is an example of how to create a network consistent with the above example configuration options, as root:

```
# nova network-create example-net --fixed-range-v4=172.16.169.0/24 --vlan=169  
--bridge=br169 --project-id=a421ae28356b4cc3a25e1429a0b02e98
```

This creates a network called `example-net` associated with tenant `a421ae28356b4cc3a25e1429a0b02e98`. The subnet is `172.16.169.0/24` with a VLAN tag of 169 (the VLAN tag does not need to match the third byte of the address, though it is a useful convention to remember the association). This will create a bridge interface device called `br169` on the host running the nova-network service. This device will appear in the output of an `ifconfig` command.

Each network is associated with one tenant. As in the example above, you may (optionally) specify this association at network creation time by using the `--project_id` flag which corresponds to the tenant ID. Use the `keystone tenant-list` command to list the tenants and corresponding IDs that you have already created.

The `nova network-create` command supports many configuration options, which are displayed when called with the `nova help network-create`:

```
usage: nova network-create [--fixed-range-v4 <x.x.x.x/yy>]  
                           [--fixed-range-v6 CIDR_V6] [--vlan <vlan id>]  
                           [--vpn <vpn start>] [--gateway GATEWAY]  
                           [--gateway-v6 GATEWAY_V6] [--bridge <bridge>]
```

```
[--bridge-interface <bridge interface>]
[--multi-host <'T' | 'F'>] [--dns1 <DNS Address>]
[--dns2 <DNS Address>] [--uuid <network uuid>]
[--fixed-cidr <x.x.x.x/yy>]
[--project-id <project id>] [--priority <number>]
<network_label>

Create a network.

Positional arguments:
<network_label>           Label for network

Optional arguments:
--fixed-range-v4 <x.x.x.x/yy>
                      IPv4 subnet (ex: 10.0.0.0/8)
--fixed-range-v6 CIDR_V6
                      IPv6 subnet (ex: fe80::/64
--vlan <vlan id>        vlan id
--vpn <vpn start>        vpn start
--gateway GATEWAY         gateway
--gateway-v6 GATEWAY_V6
                      ipv6 gateway
--bridge <bridge>        VIFs on this network are connected to this bridge
--bridge-interface <bridge interface>
                      the bridge is connected to this interface
--multi-host <'T' | 'F'>
                      Multi host
--dns1 <DNS Address>    First DNS
--dns2 <DNS Address>    Second DNS
--uuid <network uuid>
                      Network UUID
--fixed-cidr <x.x.x.x/yy>
                      IPv4 subnet for fixed IPS (ex: 10.20.0.0/16)
--project-id <project id>
                      Project id
--priority <number>       Network interface priority
```

In particular, flags to the **nova network-create** command can be used to override settings from `nova.conf`:

```
--bridge_interface  Overrides the vlan_interface configuration option
```

To view a list of the networks that have been created, as root:

```
# nova network-list
```

The **nova** command-line tool does not yet support network modifications. To modify an existing network, you must use the **nova-manage** command. Use the **nova-manage network modify** command, as root:

```
# nova-manage network modify --help
Usage: nova-manage network modify <args> [options]

Options:
-h, --help                  show this help message and exit
--fixed_range=<x.x.x.x/yy>
                           Network to modify
--project=<project name>
                           Project name to associate
```

```
--host=<host>          Host to associate
--disassociate-project
                         Disassociate Network from Project
--disassociate-host     Disassociate Host from Project
```

The **nova** command-line tool does not yet support network deletions.. To delete an existing network, you must use the **nova-manage** command. To delete a network, use **nova-manage network delete**, as root:

```
# nova-manage network delete --help
Usage: nova-manage network delete <args> [options]

Options:
-h, --help           show this help message and exit
--fixed_range=<x.x.x.x/yy>
                     Network to delete
--uuid=<uuid>       UUID of network to delete
```

Note that a network must first be disassociated from a project using the **nova network-disassociate** command before it can be deleted.

Creating a network will automatically cause the Compute database to populate with a list of available fixed IP addresses. You can view the list of fixed IP addresses and their associations with active virtual machines by doing, as root:

```
# nova-manage fix list
```

If users need to access the instances in their project across a VPN, a special VPN instance (code named **cloudpipe**) needs to be created as described in the section titled [Cloudpipe – Per Project VPNs](#).

## Libvirt VLAN networking

To configure your nodes to support VLAN tagging, install the **vlan** package and load the **8021q** kernel module, as root:

```
# apt-get install vlan
# modprobe 8021q
```

To have this kernel module loaded on boot, add the following line to `/etc/modules`:

```
8021q
```

Here is an example of settings from `/etc/nova/nova.conf` for a host configured to run **nova-network** in VLAN mode

```
network_manager=nova.network.manager.VlanManager
vlan_interface=eth0
fixed_range=172.16.0.0/12
network_size=256
```

In certain cases, the network manager may not properly tear down bridges and VLANs when it is stopped. If you attempt to restart the network manager and it does not start, check the logs for errors indicating that a bridge device already exists. If this is the case, you will likely need to tear down the bridge and VLAN devices manually. It is also advisable to

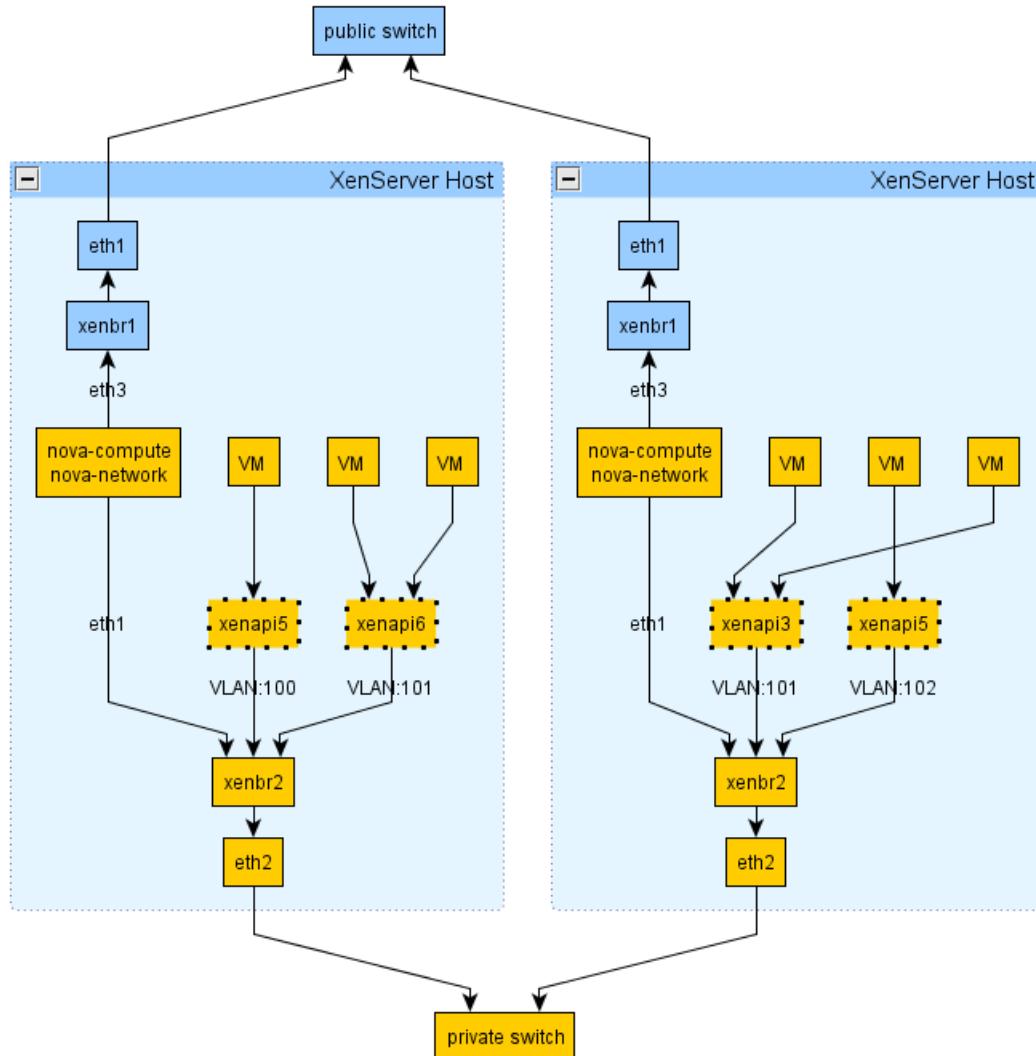
kill any remaining dnsmasq processes. These commands would stop the service, manually tear down the bridge and VLAN from the previous example, kill any remaining dnsmasq processes, and start the service up again, as root:

```
# stop nova-network
# vconfig rem vlan169
# ip link set br169 down
# brctl delbr br169
# killall dnsmasq
# start nova-network
```

## XenAPI VLAN networking

VLAN networking works quite differently with the XenAPI driver, compared to the libvirt driver. The following figure shows how your setup might look:

**Figure 10.8. VLAN network, multiple interfaces, multiple servers, network HA with XenAPI driver**



Here is an extract from a `nova.conf` file in a system running the above setup:

```
network_manager=nova.network.manager.VlanManager
xenapi_vif_driver=nova.virt.xenapi.vif.(XenAPIBridgeDriver or
    XenAPIOpenVswitchDriver)
vlan_interface=eth1
public_interface=eth3
multi_host=True
force_dhcp_release=True
send_arp_for_ha=True
flat_injected=False
firewall_driver=nova.virt.xenapi.firewall.Dom0IptablesFirewallDriver
```

You should notice that `vlan_interface` refers to the network interface on the Hypervisor and the network interface on the VM running the OpenStack services. As with before `public_interface` refers to the network interface on the VM running the OpenStack services.

With VLAN networking and the XenAPI driver, the following things happen when you start a VM:

- First the XenServer network is attached to the appropriate physical interface (PIF) and VLAN unless the network already exists.
- When the VM is created, its VIF is attached to the above network.
- The 'Openstack domU', i.e. where nova-network is running, acts as a gateway and DHCP for this instance. The DomU does this for multiple VLAN networks, so it has to be attached on a VLAN trunk. For this reason it must have an interface on the parent bridge of the VLAN bridge where VM instances are plugged.

To help understand VLAN networking with the XenAPI further, here are some important things to note:

- A physical interface (PIF) identified either by (A) the `vlan_interface` flag or (B) the `bridge_interface` column in the `networks` db table will be used for creating a XenServer VLAN network. The VLAN tag is found in the `vlan` column, still in the `networks` table, and by default the first tag is 100.
- VIF for VM instances within this network will be plugged in this VLAN network. You won't see the bridge until a VIF is plugged in it.
- The 'Openstack domU', i.e. the VM running the nova network node, instead will not be plugged into this network; since it acts as a gateway for multiple VLAN networks, it has to be attached on a VLAN trunk. For this reason it must have an interface on the parent bridge of the VLAN bridge where VM instances are plugged. For example, if `vlan_interface` is `eth0` it must be plugged in `xenbr1`, `eth1 -> xenbr1`, etc.
- Within the Openstack domU, 'ip link' is then used to configure VLAN interfaces on the 'trunk' port. Each of this vlan interfaces is associated with a `dnsmasq` instance, which will distribute IP addresses to instances. The lease file for `dnsmasq` is constantly updated by `nova-network`, thus ensuring VMs get the IP address specified by the layer3 network driver (nova IPAM or Melange).

With this configuration, VM instances should be able to get the IP address assigned to them from the appropriate dnsmasq instance, and should be able to communicate without any problem with other VMs on the same network and with their gateway.

The above point (3) probably needs some more explanations. With Open vSwitch, we don't really have distinct bridges for different VLANs; even if they appear as distinct bridges to linux and XenServer, they are actually the same OVS instance, which runs a distinct 'fake-bridge' for each VLAN. The 'real' bridge is the 'parent' of the fake one. You can easily navigate fake and real bridges with ovs-vsctl.

As you can see I am referring to Openvswitch only. This is for a specific reason: the fake-parent mechanism automatically imply that ports which are not on a fake bridge are trunk ports. This does not happen with linux bridge. A packet forwarded on a VLAN interfaces does not get back in the xenbrX bridge for ethX. For this reason, with XenAPI, you must use Open vSwitch when running VLAN networking with network HA (i.e. mult-host) enabled. On XenServer 6.0 and later, Open vSwitch is the default network stack. When using VLAN networking with XenAPI and linux bridge, the default networking stack on XenServer prior to version 6.0, you must run the network node on a VM on a XenServer that does not host any nova-compute controlled instances.

## Known issue with failed DHCP leases in VLAN configuration

Text in this section was adapted from [an email from Vish Ishaya on the OpenStack mailing list](#).

There is an issue with the way Compute uses [dnsmasq](#) in VLAN mode. Compute starts up a single copy of dnsmasq for each VLAN on the network host (or on every host in multi\_host mode). [The problem](#) is in the way that dnsmasq binds to an IP address and port. Both copies can respond to broadcast packets, but unicast packets can only be answered by one of the copies.

As a consequence, guests from only one project will get responses to their unicast DHCP renew requests. Unicast projects from guests in other projects get ignored. What happens next is different depending on the guest OS. Linux generally will send a broadcast packet out after the unicast fails, and so the only effect is a small (tens of ms) hiccup while the interface is reconfigured. It can be much worse than that, however. There have been observed cases where Windows just gives up and ends up with a non-configured interface.

This bug was first noticed by some users of OpenStack who rolled their own fix. In short, on Linux, if you set the SO\_BINDTODEVICE socket option, it will allow different daemons to share the port and respond to unicast packets, as long as they listen on different interfaces. Simon Kelley, the maintainer of dnsmasq, [has integrated a fix](#) for the issue in dnsmasq version 2.61.

If upgrading dnsmasq is out of the question, a possible workaround is to minimize lease renewals with something like the following combination of config options.

```
# release leases immediately on terminate
force_dhcp_release
# one week lease time
dhcp_lease_time=604800
# two week disassociate timeout
fixed_ip_disassociate_timeout=1209600
```

## Cloudpipe — Per Project Vpns

Cloudpipe is a method for connecting end users to their project instances in VLAN networking mode.

The support code for cloudpipe implements admin commands (via an extension) to automatically create a VM for a project that allows users to VPN into the private network of their project. Access to this VPN is provided through a public port on the network host for the project. This allows users to have free access to the virtual machines in their project without exposing those machines to the public internet.

The cloudpipe image is basically just a Linux instance with openvpn installed. It needs a simple script to grab user data from the metadata server, b64 decode it into a zip file, and run the autorun.sh script from inside the zip. The autorun script will configure and run openvpn to run using the data from nova.

It is also useful to have a cron script that will periodically redownload the metadata and copy the new Certificate Revocation List (CRL). This list is contained within the payload file and will keeps revoked users from connecting and will disconnect any users that are connected with revoked certificates when their connection is renegotiated (every hour). (More infos about revocation can be found in the following section : "Certificates and Revocation").

In this how-to, we are going to create our cloud-pipe image from a running Ubuntu instance which will serve as a template. When all the components will be installed and configured, we will create an image from that instance that will be uploaded to the Glance repositories.

### Creating a Cloudpipe Image Template

#### 1. Installing the required packages

We start by installing the required packages on our instance :

```
# apt-get update && apt-get upgrade && apt-get install openvpn bridge-utils  
unzip -y
```

#### 2. Creating the server configuration template

Create a configuration for Openvpn, and save it under /etc/openvpn/server.conf :

```
port 1194  
proto udp  
dev tap0  
up "/etc/openvpn/up.sh br0"  
down "/etc/openvpn/down.sh br0"  
script-security 3 system  
  
persist-key  
persist-tun  
  
ca ca.crt  
cert server.crt  
key server.key # This file should be kept secret
```

```
dh dh1024.pem
ifconfig-pool-persist ipp.txt

server-bridge VPN_IP DHCP_SUBNET DHCP_LOWER DHCP_UPPER

client-to-client
keepalive 10 120
comp-lzo

max-clients 1

user nobody
group nogroup

persist-key
persist-tun

status openvpn-status.log

verb 3
mute 20
```

### 3. Create the network scripts

The next step is to create both scripts that will be used when the network components will start up and shut down. The scripts will be respectively saved under /etc/openvpn/up.sh and /etc/openvpn/down.sh :

```
/etc/openvpn/up.sh

#!/bin/sh
# Openvpn startup script.

BR=$1
DEV=$2
MTU=$3
/sbin/ifconfig $DEV mtu $MTU promisc up
/sbin/brctl addif $BR $DEV
```

```
/etc/openvpn/down.sh
```

```
#!/bin/sh
# Openvpn shutdown script
BR=$1
DEV=$2

/usr/sbin/brctl delif $BR $DEV
/sbin/ifconfig $DEV down
```

Make these two scripts executables by running the following command :

```
# chmod +x /etc/openvpn/{up.sh,down.sh}
```

### 4. Edit the network interface configuration file

Update the /etc/network/interfaces accordingly (We tear down the main interface and enable the bridged interface) :

```
# This file describes the network interfaces available on your system
```

```
# and how to activate them. For more information, see interfaces(5).

# The loopback network interface
auto lo
iface lo inet loopback

# The primary network interface
auto eth0
iface eth0 inet manual
    up ifconfig $IFACE 0.0.0.0 up
    down ifconfig $IFACE down

auto br0
iface br0 inet dhcp
bridge_ports eth0
```

## 5. Edit the rc.local file

The next step consists in updating the /etc/rc.local file. We will ask our image to retrieve the payload, decrypt it, and use both key and CRL for our Openvpn service : /etc/rc.local

```
#!/bin/sh -e
#
# rc.local
#
# This script is executed at the end of each multiuser runlevel.
# Make sure that the script will "exit 0" on success or any other
# value on error.
#
# In order to enable or disable this script just change the execution
# bits.
#
# By default this script does nothing.
##### These lines go at the end of /etc/rc.local #####
. /lib/lsb/init-functions

echo Downloading payload from userdata
wget http://169.254.169.254/latest/user-data -O /tmp/payload.b64
echo Decrypting base64 payload
openssl enc -d -base64 -in /tmp/payload.b64 -out /tmp/payload.zip

mkdir -p /tmp/payload
echo Unzipping payload file
unzip -o /tmp/payload.zip -d /tmp/payload/

# if the autorun.sh script exists, run it
if [ -e /tmp/payload/autorun.sh ]; then
    echo Running autorun.sh
    cd /tmp/payload
    chmod 700 /etc/openvpn/server.key
    sh /tmp/payload/autorun.sh
    if [ ! -e /etc/openvpn/dh1024.pem ]; then
        openssl dhparam -out /etc/openvpn/dh1024.pem 1024
    fi
else
    echo rc.local : No autorun script to run
fi
```

```
exit 0
```

The called script (`autorun.sh`) is a script which mainly parses the network settings of the running instances in order to set up the initial routes. Your instance is now ready to be used as a cloudpipe image. In the next step, we will update that instance to Glance.

## Upload your instance to Glance

We will make use of the `nova` snapshot feature in order to create an image from our running instance. We start by retrieving the instance ID :

```
$ nova list
```

	ID	Name	Status	Networks
	739079ab-0f8e-404a-ae6e-a91f4fe99c94	cloud-pipe	ACTIVE	vlan1=192.168.22.43

We create an image with, using the instance ID :

```
$ nova image-create 739079ab-0f8e-404a-ae6e-a91f4fe99c94
```

Make sure the instance has been upload to the Glance repository :

```
$ nova image-list
```

Server	ID	Name	Status
	0bfc8fd3-1590-463b-b178-bce30be5ef7b	cloud-pipance	ACTIVE

Make that image public (snapshot-based images are private by default):

```
$ glance image-update 0bfc8fd3-1590-463b-b178-bce30be5ef7b is_public=true
```

You can ensure the image is now public, running

```
$ glance show 0bfc8fd3-1590-463b-b178-bce30be5ef7b | grep Public
```

Public : Yes

## Update /etc/nova.conf

Some settings need to be added into `/etc/nova.conf` file in order to make `nova` able to use our image : `/etc/nova.conf`

```
## cloud-pipe vpn client ##
vpn_image_id=0bfcc8fd3-1590-463b-b178-bce30be5ef7b
use_project_ca=true
cnt_vpn_clients=5
```

You can now restart all the services :

```
# cd /etc/int.d && for i in $( ls nova-* ); do service $i restart; done
```

## Power-up your instance

Use the nova cloudpipe feature the following way :

```
$ nova cloudpipe create $tenant_id
```

Retrive all the tenants :

```
$ keystone tenant-list
```

id	name	enabled
071ffb95837e4d509cb7153f21c57c4d	stone	True
520b6689e344456cbb074c83f849914a	service	True
d1f5d27ccf594cdbb034c8a4123494e9	admin	True
dfb0ef4ab6d94d5b9e9e0006d0ac6706	demo	True

Let's create our cloudpipe project using the tenant's ID :

```
$ nova cloudpipe-create d1f5d27ccf594cdbb034c8a4123494e9
```

We can check the service availability :

```
$ nova cloudpipe-list
```

Project Id	Public IP	Public Port	Internal IP
d1f5d27ccf594cdbb034c8a4123494e9	172.17.1.3	1000	192.168.22.34

The output basically shows our instance is started. Nova will create the necessary rules for our cloudpipe instance (icmp and OpenVPN port) :

```
ALLOW 1194:1194 from 0.0.0.0/0
ALLOW -1:-1 from 0.0.0.0/0
```

## VPN Access

In VLAN networking mode, the second IP in each private network is reserved for the cludpipe instance. This gives a consistent IP to the instance so that nova-network can create forwarding rules for access from the outside world. The network for each project is given a specific high-numbered port on the public IP of the network host. This port is automatically forwarded to 1194 on the VPN instance.

If specific high numbered ports do not work for your users, you can always allocate and associate a public IP to the instance, and then change the `vpn_public_ip` and `vpn_public_port` in the database. Rather than using the database directly, you can also use `nova-manage vpn change [new_ip] [new_port]`

## Certificates and Revocation

For certificate management, it is also useful to have a cron script that will periodically download the metadata and copy the new Certificate Revocation List (CRL). This will keep revoked users from connecting and disconnects any users that are connected with revoked certificates when their connection is re-negotiated (every hour). You set the `use_project_ca` option in `nova.conf` for cludpipes to work securely so that each project has its own Certificate Authority (CA).

If the `use_project_ca config` option is set (required to for cludpipes to work securely), then each project has its own CA. This CA is used to sign the certificate for the vpn, and is also passed to the user for bundling images. When a certificate is revoked using `nova-manage`, a new Certificate Revocation List (crl) is generated. As long as cludpipe has an updated crl, it will block revoked users from connecting to the vpn.

The userdata for cludpipe isn't currently updated when certs are revoked, so it is necessary to restart the cludpipe instance if a user's credentials are revoked.

## Restarting and Logging into the Cludpipe VPN

You can reboot a cludpipe vpn through the api if something goes wrong (using `nova reboot` for example), but if you generate a new crl, you will have to terminate it and start it again using the cludpipe extension. The cludpipe instance always gets the first ip in the subnet and if `force_dhcp_release` is not set it takes some time for the ip to be recovered. If you try to start the new vpn instance too soon, the instance will fail to start because of a "NoMoreAddresses" error. It is therefore recommended to use `force_dhcp_release`.

The keypair that was used to launch the cludpipe instance should be in the `keys/<project_id>` folder. You can use this key to log into the cludpipe instance for debugging purposes. If you are running multiple copies of `nova-api` this key will be on whichever server used the original request. To make debugging easier, you may want to put a common administrative key into the cludpipe image that you create.

## Remote access to your cludpipe instance from an OpenVPN client

Now your cludpipe instance is running, you can use your favorite OpenVPN client in order to access your instances within their private network cludpipe is connected to. In these

sections we will present both ways of using cloupipe, the first using a configuration file for clients without interfaces, and for clients using an interface.

### Connect to your cloupipe instance without an interface (CLI)

#### 1. Generate your certificates

Start by generating a private key and a certificate for your project:

```
$ nova x509-create-cert
```

#### 2. Create the openvpn configuration file

The following template, which can be found under `nova/cloupipe/client.ovpn.template` contains the necessary instructions for establishing a connection :

```
# NOVA user connection
# Edit the following lines to point to your cert files:
cert /path/to/the/cert/file
key /path/to/the/key/file

ca cacert.pem

client
dev tap
proto udp

remote $cloupipe-public-ip $cloupipe-port
resolv-retry infinite
nobind

# Downgrade privileges after initialization (non-Windows only)
user nobody
group nogroup
comp-lzo

# Set log file verbosity.
verb 2

keepalive 10 120
ping-timer-rem
persist-tun
persist-key
```

Update the file accordingly. In order to get the public IP and port of your cloupipe instance, you can run the following command :

```
$ nova cloupipe-list
```

Project Id	Public IP	Public Port	Internal IP
d1f5d27ccf594cd8b034c8a4123494e9 34	172.17.1.3	1000	192.168.22.

### 3. Start your OpenVPN client

Depending on the client you are using, make sure to save the configuration file under the directory it should be, so the certificate file and the private key. Usually, the file is saved under /etc/openvpn/clientconf/client.conf

#### Connect to your cloupipe instance using an interface

##### 1. Download an OpenVPN client

In order to connect to the project's network, you will need an OpenVPN client for your computer. Here are several clients

- For Ubuntu :

[OpenVPN](#)

[network-manager-openvpn](#)

[kvpnc](#) (For Kubuntu)

[gopenvpn](#)

- For Mac OsX :

[OpenVPN \(Official Client\)](#)

[Viscosity](#)

[Tunnelblick](#)

- For Windows :

[OpenVPN \(Official Client\)](#)

##### 2. Configure your client

In this example we will use Viscosity, but the same settings apply to any client. Start by filling the public ip and the public port of the cloupipe instance.

These informations can be found by running a

```
$ nova cloupipe-list
```

Project Id	Public IP	Public Port	Internal IP
d1f5d27ccf594cd8b034c8a4123494e9 34	172.17.1.3	1000	192.168.22.

**Figure 10.9. Configuring Viscosity**

- Connection Name : "Openstack-cloudpipe"

Remote server : "172.17.1.3"

Port : "1000"

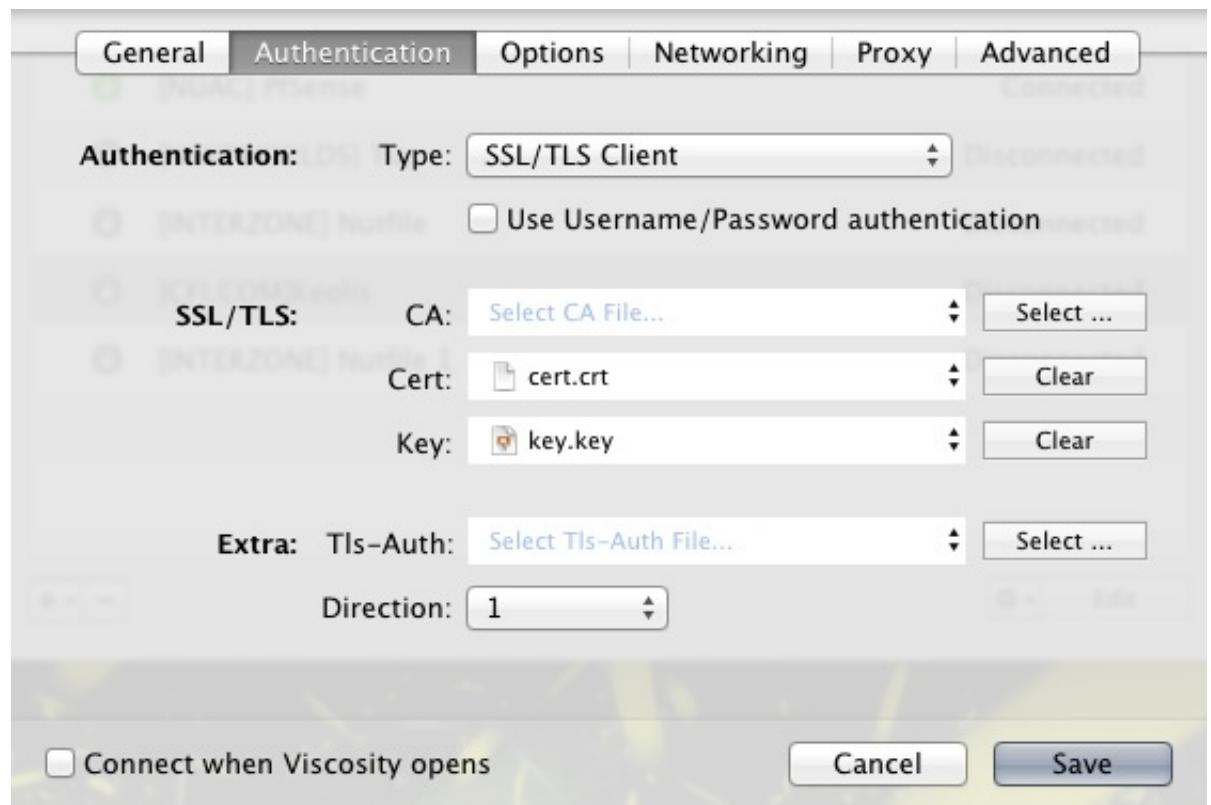
Protocol : "udp"

Device Type : "tap"



- Certificate : The generated certificate

Key : The private key



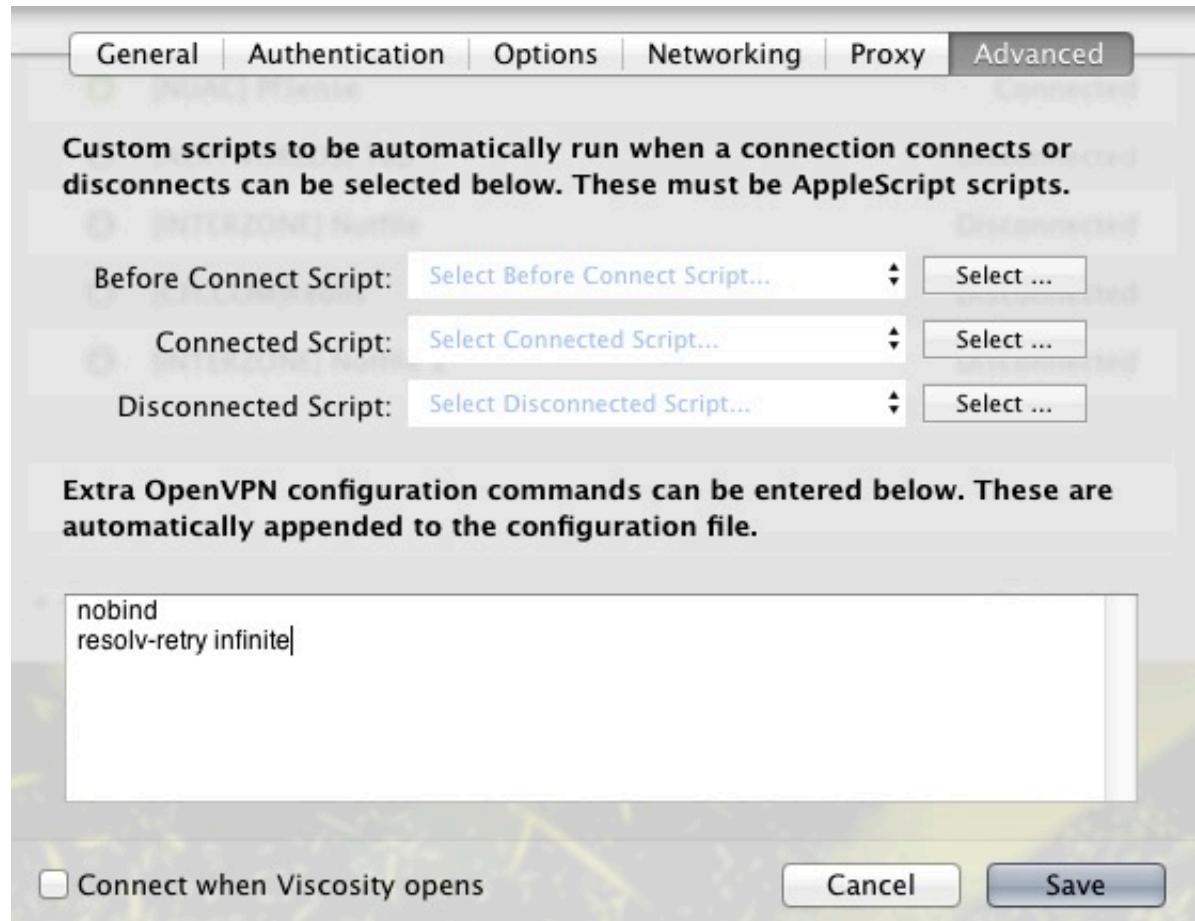
- Persistence options : "Persistent TUN" and "Persistent key"

Other :" No bind"



- Advanced

Extra settings : "nobind" and "resolv-retry infinite"



You can now save the configuration and establish the connection!

## Cloudpipe Troubleshooting and Automation

- **Troubleshoot your cloudpipe instance**

A periodic task disassociates the fixed ip address for the cloudpipe instance. Into /var/log/nova/nova-network.log, the following line should appear :

```
Running periodic task VlanManager._disassociate_stale_fixed_ips from (pid=21578) periodic_tasks /usr/lib/python2.7/dist-packages/nova/manager.py:152
```

Once the job has been run, \$ **nova cloudpipe-list** should not return anything ; but if the cloudpipe instance is respawned too quickly; the following error could be encountered :

```
ERROR nova.rpc.amqp Returning exception Fixed IP address 192.168.22.34 is already in use.
```

In order to resolve that issue, log into the mysql server and update the ip address status :

```
(mysql) use nova;
(mysql) SELECT * FROM fixed_ips WHERE address='192.168.22.34';

+-----+-----+-----+-----+
| created_at | updated_at | deleted_at | deleted | | |
| address | network_id | instance_id | allocated | leased | reserved |
| virtual_interface_id | host |
+-----+-----+-----+-----+
| 2012-05-21 12:06:18 | 2012-06-18 09:26:25 | NULL | 0 | 0 | 484 |
| 192.168.22.34 | 13 | 630 | 0 | 0 | 1 |
| NULL | NULL |
+-----+-----+-----+-----+
+-----+-----+-----+-----+
+-----+-----+-----+-----+
| created_at | updated_at | deleted_at | deleted | | | |
| id | address | network_id | instance_id | allocated | leased | reserved |
| virtual_interface_id | |
+-----+-----+-----+-----+
| 2012-05-21 12:06:18 | 2012-06-18 09:26:25 | NULL | 0 | 0 | 484 |
| 192.168.22.34 | 13 | NULL | 0 | 0 | 1 |
| NULL | NULL |
+-----+-----+-----+-----+
```

- **Cloudpipe-related configuration option reference**

```
vpn_ip = COMPUTE_NODE_IP
vpn_start = 1000
vpn_key_suffix = -vpn
vpn_client_template = /usr/lib/python2.7/dist-packages/nova/cloudpipe/
client.ovpn.template
credential_vpn_file = nova-vpn.conf
vpn_image_id = IMAGE_ID
cnt_vpn_clients = 5
keys_path = /var/lib/nova/keys
ca_path = /var/lib/nova/CA
```

- **Cloudpipe-related files**

Nova stores cloudpipe keys into /var/lib/nova/keys.

Certificates are stored into /var/lib/nova/CA.

Credentials are stored into /var/lib/nova/CA/projects/

- **Automate the cloudpipe image installation**

You can automate the image creation by download that script and running it from inside the instance : [Get the script from Github](#)

## Enabling Ping and SSH on VMs

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



### Note

These commands need to be run as root only if the credentials used to interact with nova-api have been put under /root/.bashrc. If the EC2 credentials have been put into another user's .bashrc file, then, it is necessary to run these commands as the user.

Using the nova command-line tool:

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

Using euca2ools:

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

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

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

# Configuring Public (Floating) IP Addresses

## Private and Public IP Addresses

Every virtual instance is automatically assigned a private IP address. You may optionally assign public IP addresses to instances. OpenStack uses the term "floating IP" to refer to an IP address (typically public) that can be dynamically added to a running virtual instance. OpenStack Compute uses Network Address Translation (NAT) to assign floating IPs to virtual instances.

If you plan to use this feature, you must add the following to your nova.conf file to specify which interface the nova-network service will bind public IP addresses to:

```
public_interface=vlan100
```

Restart the nova-network service if you change nova.conf while the service is running.



### Traffic between VMs using floating IPs

Note that due to the way floating IPs are implemented using a source NAT (SNAT rule in iptables), inconsistent behaviour of security groups can be seen if VMs use their floating IP to communicate with other virtual machines - particularly on the same physical host. Traffic from VM to VM across the fixed network does not have this issue, and this is the recommended path. To ensure traffic doesn't get SNATed to the floating range, explicitly set `dmz_cidr=x.x.x.x/y`. `x.x.x.x/y` is the range of floating ips for each pool of floating ips you define. This configuration is also necessary to make `source_groups` work if the vms in the source group have floating ips.

## Enabling IP forwarding

By default, the IP forwarding is disabled on most of Linux distributions. The "floating IP" feature requires the IP forwarding enabled in order to work.



### Note

The IP forwarding only needs to be enabled on the nodes running the service nova-network. If the `multi_host` mode is used, make sure to enable it on all the compute node, otherwise, enable it on the node running the nova-network service.

you can check if the forwarding is enabled by running the following command:

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

```
0
```

Or using sysctl

```
$ sysctl net.ipv4.ip_forward
```

```
net.ipv4.ip_forward = 0
```

In this example, the IP forwarding is disabled. You can enable it on the fly by running the following command:

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

or

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

In order to make the changes permanent, edit the `/etc/sysctl.conf` and update the IP forwarding setting :

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

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

```
$ sysctl -p
```

It is also possible to update the setting by restarting the network service. Here's an example for Ubuntu:

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

Here's an example for RHEL/Fedora/CentOS:

```
$ service network restart
```

## Creating a List of Available Floating IP Addresses

Nova 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 this list, as root.

For example:

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

The following nova-manage commands apply to floating IPs.

- **nova-manage floating list**: List the floating IP addresses in the pool.
- **nova-manage floating create --pool=[pool name] --ip\_range=[CIDR]**: Create specific floating IPs for either a single address or a subnet.
- **nova-manage floating delete [cidr]**: Remove floating IP addresses using the same parameters as the create command.

## Adding a Floating IP to an Instance

Adding a floating IP to an instance is a two step process:

1. **nova floating-ip-create**: Allocate a floating IP address from the list of available addresses.

2. **nova add-floating-ip**: Add an allocated floating IP address to a running instance.

Here's an example of how to add a floating IP to a running instance with an ID of 12

```
$ nova floating-ip-create

+-----+-----+-----+-----+
|     Ip      | Instance Id | Fixed Ip | Pool |
+-----+-----+-----+-----+
| 68.99.26.170 | None        | None      |      |
+-----+-----+-----+-----+

$ nova add-floating-ip 12 68.99.26.170
```

If the instance no longer needs a public address, remove the floating IP address from the instance and de-allocate the address:

```
$ nova remove-floating-ip 12 68.99.26.170
$ nova floating-ip-delete 68.99.26.170
```

## Automatically adding floating IPs

The nova-network service can be configured to automatically allocate and assign a floating IP address to virtual instances when they are launched. Add the following line to `nova.conf` and restart the nova-network service

```
auto_assign_floating_ip=True
```

Note that if this option is enabled and all of the floating IP addresses have already been allocated, the **nova boot** command will fail with an error.

## Removing a Network from a Project

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

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

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

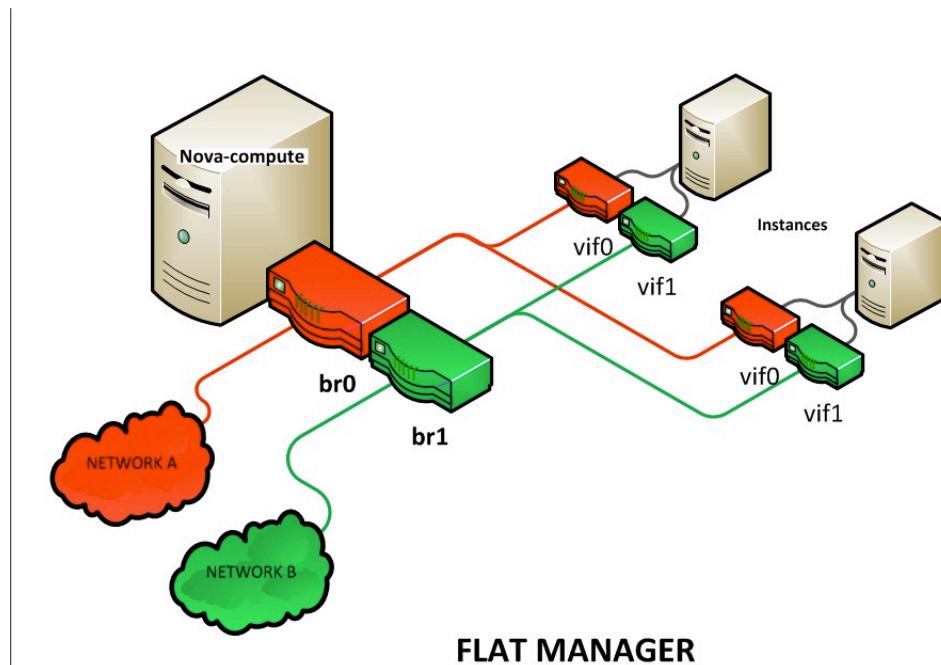
## Using multiple interfaces for your instances (multinic)

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

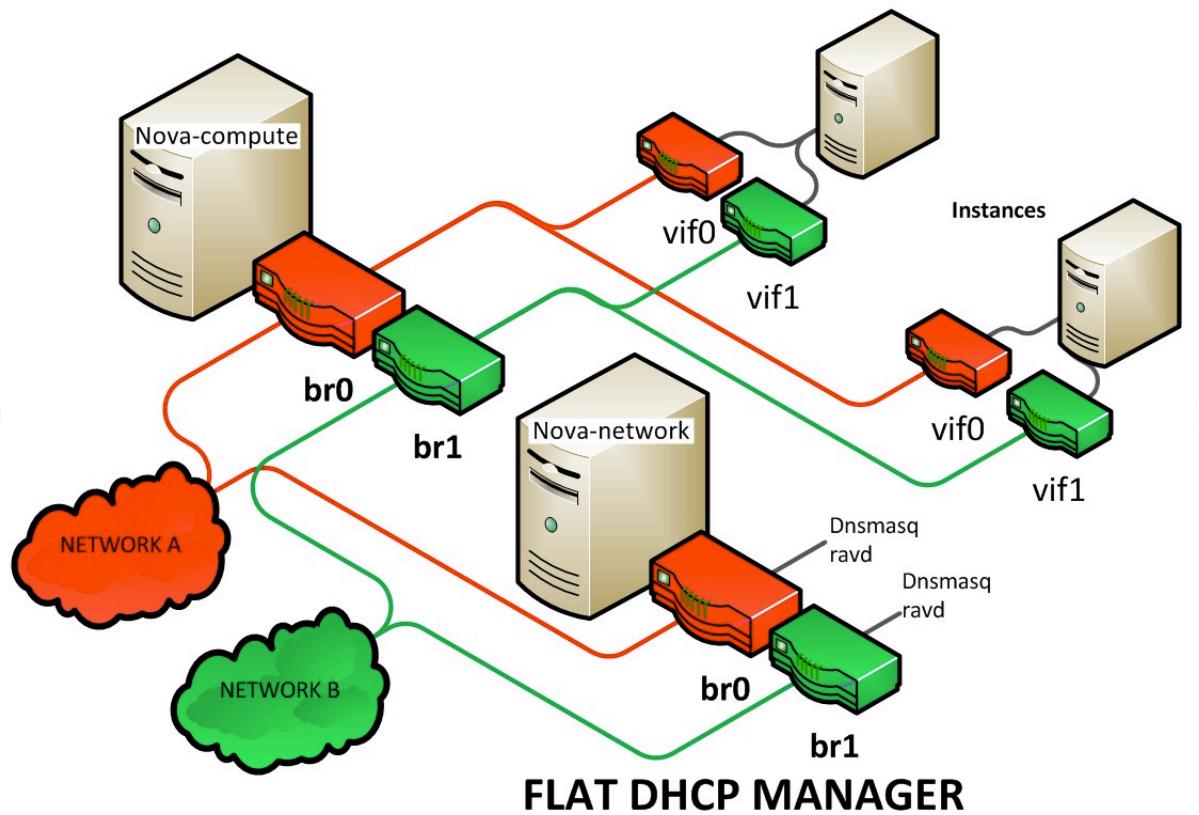
- SSL Configurations (VIPs)
- Services failover/ HA
- Bandwidth Allocation
- Administrative/ Public access to your instances

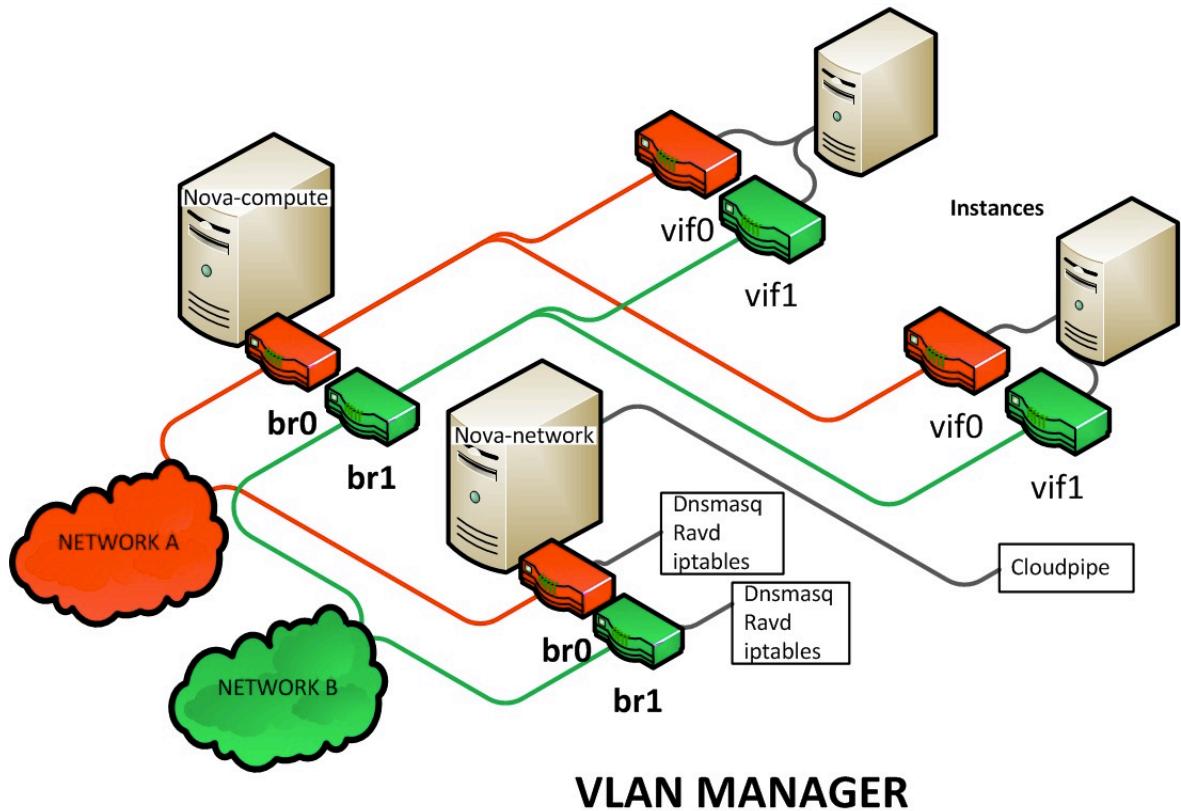
Each VIF is representative of a separate network with its own IP block. Every network mode introduces it's own set of changes regarding the mulitnic usage :

**Figure 10.10. multicnic flat manager**



**Figure 10.11. multinic flatdhcp manager**



**Figure 10.12. multinic VLAN manager**

## Using the multinic feature

In order to use the multinic feature, first create two networks, and attach them to your project :

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

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

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



### Note

Make sure to power up the second interface on the instance, otherwise that last won't be reachable via its second IP. Here is an example of how to setup

the interfaces within the instance (this is the configuration that needs to be applied inside the image) :

```
/etc/network/interfaces

# The loopback network interface
auto lo
iface lo inet loopback

auto eth0
iface eth0 inet dhcp

auto eth1
iface eth1 inet dhcp
```

### Note

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

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

## Existing High Availability Options for Networking

Based off a blog post by [Vish Ishaya](#)

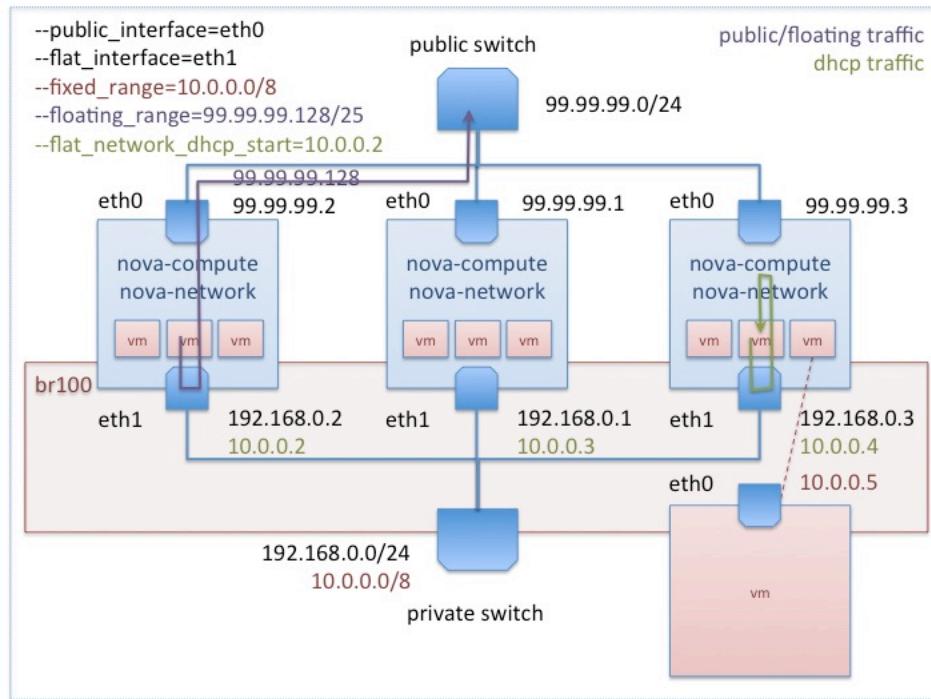
As illustrated in the Flat DHCP diagram in Section [Configuring Flat DHCP Networking](#) titled [Flat DHCP network, multiple interfaces, multiple servers](#), traffic from the VM to the public internet has to go through the host running `nova` network. DHCP is handled by `nova-network` as well, listening on the gateway address of the `fixed_range` network. The compute hosts can optionally have their own public IPs, or they can use the network host as their gateway. This mode is pretty simple and it works in the majority of situations, but it has one major drawback: the network host is a single point of failure! If the network host goes down for any reason, it is impossible to communicate with the VMs. Here are some options for avoiding the single point of failure.

### HA Option 1: Multi-host

To eliminate the network host as a single point of failure, Compute can be configured to allow each compute host to do all of the networking jobs for its own VMs. Each compute host does NAT, DHCP, and acts as a gateway for all of its own VMs. While there is still a single point of failure in this scenario, it is the same point of failure that applies to all virtualized systems.

This setup requires adding an IP on the VM network to each host in the system, and it implies a little more overhead on the compute hosts. It is also possible to combine this with option 4 (HW Gateway) to remove the need for your compute hosts to gateway. In that hybrid version they would no longer gateway for the VMs and their responsibilities would only be DHCP and NAT.

The resulting layout for the new HA networking option looks the following diagram:

**Figure 10.13. High Availability Networking Option**

In contrast with the earlier diagram, all the hosts in the system are running the nova-compute, nova-network and nova-api services. Each host does DHCP and does NAT for public traffic for the VMs running on that particular host. In this model every compute host requires a connection to the public internet and each host is also assigned an address from the VM network where it listens for DHCP traffic. The nova-api service is needed so that it can act as a metadata server for the instances.

To run in HA mode, each compute host must run the following services:

- **nova-compute**
- **nova-network**
- **nova-api-metadata or nova-api**

If the compute host is not an API endpoint, use the **nova-api-metadata** service. The `nova.conf` file should contain:

```
multi_host=True  
send_arp_for_ha=true
```

The `send_arp_for_ha` option facilitates sending of gratuitous arp messages to ensure the arp caches on compute hosts are up to date.

If a compute host is also an API endpoint, use the **nova-api** service. Your `enabled_apis` option will need to contain `metadata`, as well as additional options depending on the API services. For example, if it supports compute requests, volume requests, and EC2 compatibility, the `nova.conf` file should contain:

```
multi_host=True
send_arp_for_ha=true
enabled_apis=ec2,osapi_compute,osapi_volume,metadata
```

The `multi_host` option must be in place when you create the network and `nova-network` must be run on every compute host. These created multi hosts networks will send all network related commands to the host that the specific VM is on. You need to edit the configuration option `enabled_apis` such that it includes `metadata` in the list of enabled APIs. Other options become available when you configure `multi_host` nova networking please refer to [Configuration: nova.conf](#).



### Note

You must specify the `multi_host` option on the command line when creating fixed networks. For example:

```
# nova network-create test --fixed-range-v4=192.168.0.0/24 --multi-
host=T
```

## HA Option 2: Failover

The folks at NTT labs came up with a ha-linux configuration that allows for a 4 second failover to a hot backup of the network host. Details on their approach can be found in the following post to the openstack mailing list: <https://lists.launchpad.net/openstack/msg02099.html>

This solution is definitely an option, although it requires a second host that essentially does nothing unless there is a failure. Also four seconds can be too long for some real-time applications.

To enable this HA option, your `nova . conf` file must contain the following option:

```
send_arp_for_ha=True
```

See <https://bugs.launchpad.net/nova/+bug/782364> for details on why this option is required when configuring for failover.

## HA Option 3: Multi-nic

Recently, `nova` gained support for multi-nic. This allows us to bridge a given VM into multiple networks. This gives us some more options for high availability. It is possible to set up two networks on separate vlans (or even separate ethernet devices on the host) and give the VMs a NIC and an IP on each network. Each of these networks could have its own network host acting as the gateway.

In this case, the VM has two possible routes out. If one of them fails, it has the option of using the other one. The disadvantage of this approach is it offloads management of failure scenarios to the guest. The guest needs to be aware of multiple networks and have a strategy for switching between them. It also doesn't help with floating IPs. One would have to set up a floating IP associated with each of the IPs on private the private networks to achieve some type of redundancy.

## HA Option 4: Hardware gateway

The `dnsmasq` service can be configured to use an external gateway instead of acting as the gateway for the VMs. This offloads HA to standard switching hardware and it has some strong benefits. Unfortunately, the `nova-network` service is still responsible for floating IP natting and DHCP, so some failover strategy needs to be employed for those options. To configure for hardware gateway:

1. Create a `dnsmasq` configuration file (e.g., `/etc/dnsmasq-nova.conf`) that contains the IP address of the external gateway. If running in FlatDHCP mode, assuming the IP address of the hardware gateway was 172.16.100.1, the file would contain the line:

```
dhcp-option=option:router,172.16.100.1
```

If running in VLAN mode, a separate router must be specified for each network. The networks are identified by the first argument when calling `nova network-create` to create the networks as documented in the [Configuring VLAN Networking subsection](#). Assuming you have three VLANs, that are labeled red, green, and blue, with corresponding hardware routers at 172.16.100.1, 172.16.101.1 and 172.16.102.1, the `dnsmasq` configuration file (e.g., `/etc/dnsmasq-nova.conf`) would contain the following:

```
dhcp-option=tag:'red',option:router,172.16.100.1
dhcp-option=tag:'green',option:router,172.16.101.1
dhcp-option=tag:'blue',option:router,172.16.102.1
```

2. Edit `/etc/nova/nova.conf` to specify the location of the `dnsmasq` configuration file:

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

3. Configure the hardware gateway to forward metadata requests to a host that's running the `nova-api` service with the metadata API enabled.

The virtual machine instances access the metadata service at 169.254.169.254 port 80. The hardware gateway should forward these requests to a host running the `nova-api` service on the port specified as the `metadata_host` config option in `/etc/nova/nova.conf`, which defaults to 8775.

Make sure that the list in the `enabled_apis` configuration option `/etc/nova/nova.conf` contains `metadata` in addition to the other APIs. An example that contains the EC2 API, the OpenStack compute API, the OpenStack volume API, and the metadata service would look like:

```
enabled_apis=ec2,osapi_compute,osapi_volume,metadata
```

4. Ensure you have set up routes properly so that the subnet that you use for virtual machines is routable.

## Troubleshooting Networking

### Can't reach floating IPs

If you aren't able to reach your instances via the floating IP address, make sure the default security group allows ICMP (ping) and SSH (port 22), so that you can reach the instances:

```
$ nova secgroup-list-rules default

+-----+-----+-----+-----+
| IP Protocol | From Port | To Port | IP Range | Source Group |
+-----+-----+-----+-----+
| icmp       | -1        | -1      | 0.0.0.0/0 |           |
| tcp        | 22        | 22      | 0.0.0.0/0 |           |
+-----+-----+-----+-----+
```

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

```
# iptables -L -nv

-A nova-network-OUTPUT -d 68.99.26.170/32 -j DNAT --to-destination 10.0.0.3

# iptables -L -nvv -t nat

-A nova-network-PREROUTING -d 68.99.26.170/32 -j DNAT --to-destination 10.0.0.3
-A nova-network-floating-snat -s 10.0.0.3/32 -j SNAT --to-source 68.99.26.170
```

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

```
$ ip addr

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

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

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

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

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

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

## Disabling firewall

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

## KVM: Network connectivity works initially, then fails

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

```
# modprobe vhost_net
```

Note that loading the module will have no effect on instances that are already running.

# 11. Volumes

The OpenStack Block Storage service provides persistent block storage resources that OpenStack Compute instances can consume.

Refer to the [OpenStack Block Storage Admin Manual](#) for information about configuring volume drivers and creating and attaching volumes to server instances.

# 12. Scheduling

## Table of Contents

Filter Scheduler .....	250
Filters .....	251
Costs and Weights .....	257
Other Schedulers .....	259
Host aggregates .....	259

Compute uses the **nova-scheduler** service to determine how to dispatch compute and volume requests. For example, the **nova-scheduler** service determines which host a VM should launch on. The term "host" in the context of filters means a physical node that has a **nova-compute** service running on it. The scheduler is configurable through a variety of options.

Compute is configured with the following default scheduler options:

```
scheduler_driver=nova.scheduler.multi.MultiScheduler
volume_scheduler_driver=nova.scheduler.chance.ChanceScheduler
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn
compute_fill_first_cost_fn_weight=-1.0
```

Compute is configured by default to use the Multi Scheduler, which allows the admin to specify different scheduling behavior for compute requests versus volume requests.

The volume scheduler is configured by default as a Chance Scheduler, which picks a host at random that has the **cinder-volume** service running.

The compute scheduler is configured by default as a Filter Scheduler, described in detail in the next section. In the default configuration, this scheduler will only consider hosts that are in the requested availability zone (`AvailabilityZoneFilter`), that have sufficient RAM available (`RamFilter`), and that are actually capable of servicing the request (`ComputeFilter`).

From the resulting filtered list of eligible hosts, the scheduler will assign a cost to each host based on the amount of free RAM (`nova.scheduler.least_cost.compute_fill_first_cost_fn`), will multiply each cost value by -1 (`compute_fill_first_cost_fn_weight`), and will select the host with the minimum cost. This is equivalent to selecting the host with the maximum amount of RAM available.

## Filter Scheduler

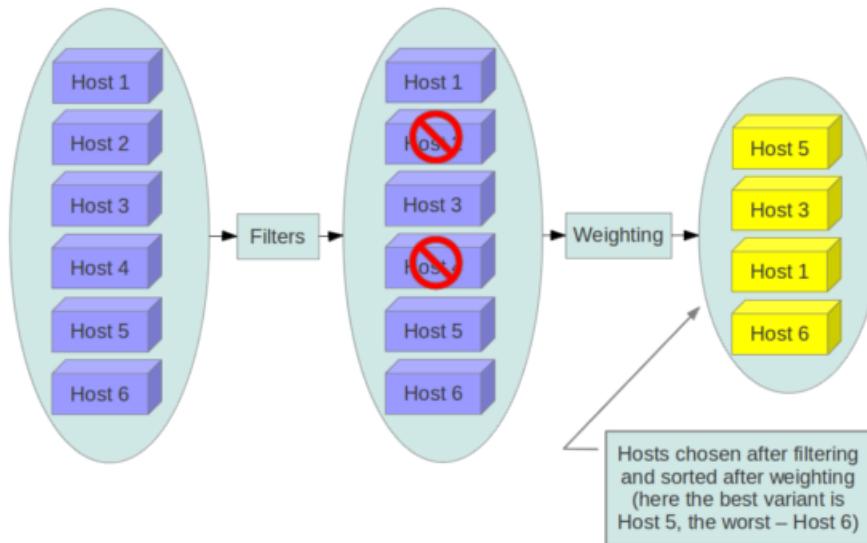
The Filter Scheduler (`nova.scheduler.filter_scheduler.FilterScheduler`) is the default scheduler for scheduling virtual machine instances. It supports filtering and weighting to make informed decisions on where a new instance should be created. This

Scheduler can only be used for scheduling compute requests, not volume requests, i.e. it can only be used with the `compute_scheduler_driver` configuration option.

## Filters

When the Filter Scheduler receives a request for a resource, it first applies filters to determine which hosts are eligible for consideration when dispatching a resource. Filters are binary: either a host is accepted by the filter, or it is rejected. Hosts that are accepted by the filter are then processed by a different algorithm to decide which hosts to use for that request, described in the [costs and weight](#) section.

**Figure 12.1. Filtering**



The `scheduler_available_filters` configuration option in `nova.conf` provides the Compute service with the list of the filters that will be used by the scheduler. The default setting specifies all of the filter that are included with the Compute service:

```
scheduler_available_filters=nova.scheduler.filters.all_filters
```

This configuration option can be specified multiple times. For example, if you implemented your own custom filter in Python called `myfilter.MyFilter` and you wanted to use both the built-in filters and your custom filter, your `nova.conf` file would contain:

```
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_available_filters=myfilter.MyFilter
```

The `scheduler_default_filters` configuration option in `nova.conf` defines the list of filters that will be applied by the `nova-scheduler` service. As mentioned above, the default filters are:

```
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter
```

The available filters are described below.

## AggregateInstanceExtraSpecsFilter

Matches properties defined in an instance type's extra specs against admin-defined properties on a host aggregate. See the [host aggregates](#) section for documentation on how to use this filter.

## AggregateMultiTenancyIsolation

Isolates tenants to specific [host aggregates](#). If a host is in an aggregate that has the metadata key `filter_tenant_id` it will only create instances from that tenant (or list of tenants). A host can be in different aggregates. If a host does not belong to an aggregate with the metadata key, it can create instances from all tenants.

## AllHostsFilter

This is a no-op filter, it does not eliminate any of the available hosts.

## AvailabilityZoneFilter

Filters hosts by availability zone. This filter must be enabled for the scheduler to respect availability zones in requests.

## ComputeCapabilitiesFilter

Matches properties defined in an instance type's extra specs against compute capabilities.

If an extra specs key contains a colon ":" , anything before the colon is treated as a namespace, and anything after the colon is treated as the key to be matched. If a namespace is present and is not 'capabilities', it is ignored by this filter.



### Note

Disable the ComputeCapabilitiesFilter when using a Bare Metal configuration, due to [bug 1129485](#)

## ComputeFilter

Filters hosts by flavor (also known as instance type) and image properties. The scheduler will check to ensure that a compute host has sufficient capabilities to run a virtual machine instance that corresponds to the specified flavor. If the image has properties specified, this filter will also check that the host can support them. The image properties that the filter checks for are:

- `architecture`: Architecture describes the machine architecture required by the image. Examples are `i686`, `x86_64`, `arm`, and `ppc64`.

- `hypervisor_type`: Hypervisor type describes the hypervisor required by the image. Examples are `xen`, `kvm`, `qemu`, `xenapi`, and `powervm`.
- `vm_mode`: Virtual machine mode describes the hypervisor application binary interface (ABI) required by the image. Examples are '`xen`' for Xen 3.0 paravirtual ABI, '`hvm`' for native ABI, '`uml`' for User Mode Linux paravirtual ABI, `exe` for container virt executable ABI.

In general, this filter should always be enabled.

## CoreFilter

Only schedule instances on hosts if there are sufficient CPU cores available. If this filter is not set, the scheduler may over provision a host based on cores (i.e., the virtual cores running on an instance may exceed the physical cores).

This filter can be configured to allow a fixed amount of vCPU overcommitment by using the `cpu_allocation_ratio` Configuration option in `nova.conf`. The default setting is:

```
cpu_allocation_ratio=16.0
```

With this setting, if there are 8 vCPUs on a node, the scheduler will allow instances up to 128 vCPU to be run on that node.

To disallow vCPU overcommitment set:

```
cpu_allocation_ratio=1.0
```

## DifferentHostFilter

Schedule the instance on a different host from a set of instances. To take advantage of this filter, the requester must pass a scheduler hint, using `different_host` as the key and a list of instance uuids as the value. This filter is the opposite of the `SameHostFilter`. Using the `nova` command-line tool, use the `--hint` flag. For example:

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 --hint different_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 --hint different_host=8c19174f-4220-44f0-824a-cd1eeef10287 server-1
```

With the API, use the `os:scheduler_hints` key. For example:

```
{
    'server': {
        'name': 'server-1',
        'imageRef': 'cedef40a-ed67-4d10-800e-17455edce175',
        'flavorRef': '1'
    },
    'os:scheduler_hints': {
        'different_host': ['a0cf03a5-d921-4877-bb5c-86d26cf818e1',
                           '8c19174f-4220-44f0-824a-cd1eeef10287'],
    }
}
```

## DiskFilter

Only schedule instances on hosts if there is sufficient disk space available for root and ephemeral storage.

This filter can be configured to allow a fixed amount of disk overcommitment by using the `disk_allocation_ratio` Configuration option in `nova.conf`. The default setting is:

```
disk_allocation_ratio=1.0
```

Adjusting this value to be greater than 1.0 will allow scheduling instances while over committing disk resources on the node. This may be desirable if you use an image format that is sparse or copy on write such that each virtual instance does not require a 1:1 allocation of virtual disk to physical storage.

## GroupAntiAffinityFilter

The GroupAntiAffinityFilter ensures that each instance in a group is on a different host. To take advantage of this filter, the requester must pass a scheduler hint, using `group` as the key and a list of instance uuids as the value. Using the `nova` command-line tool, use the `--hint` flag. For example:

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor
 1 --hint group=a0cf03a5-d921-4877-bb5c-86d26cf818e1 --hint group=
 8c19174f-4220-44f0-824a-cd1eeef10287 server-1
```

## ImagePropertiesFilter

Filters hosts based on properties defined on the instance's image. It passes hosts that can support the specified image properties contained in the instance. Properties include the architecture, hypervisor type, and virtual machine mode. E.g., an instance might require a host that runs an ARM-based processor and QEMU as the hypervisor. An image can be decorated with these properties using

```
glance image-update img-uuid --property architecture=arm --property
hypervisor_type=qemu
```

## IsolatedHostsFilter

Allows the admin to define a special (isolated) set of images and a special (isolated) set of hosts, such that the isolated images can only run on the isolated hosts, and the isolated hosts can only run isolated images.

The admin must specify the isolated set of images and hosts in the `nova.conf` file using the `isolated_hosts` and `isolated_images` configuration options. For example:

```
isolated_hosts=server1,server2
isolated_images=342b492c-128f-4a42-8d3a-c5088cf27d13,ebd267a6-ca86-4d6c-9a0e-
bd132d6b7d09
```

## JsonFilter

The JsonFilter allows a user to construct a custom filter by passing a scheduler hint in JSON format. The following operators are supported:

- =
- <
- >
- in
- <=
- >=
- not
- or
- and

The filter supports the following variables:

- \$free\_ram\_mb
- \$free\_disk\_mb
- \$total\_usable\_ram\_mb
- \$vcpus\_total
- \$vcpus\_used

Using the **nova** command-line tool, use the **--hint** flag:

```
$ nova boot --image 827d564a-e636-4fc4-a376-d36f7ebe1747 --flavor  
1 --hint query='[ ">=" , "$free_ram_mb" , 1024 ]' server1
```

With the API, use the `os:scheduler_hints` key:

```
{  
    'server': {  
        'name': 'server-1',  
        'imageRef': 'cedef40a-ed67-4d10-800e-17455edce175',  
        'flavorRef': '1'  
    },  
    'os:scheduler_hints': {  
        'query': '[ ">=" , "$free_ram_mb" , 1024 ]'  
    }  
}
```

## RamFilter

Only schedule instances on hosts if there is sufficient RAM available. If this filter is not set, the scheduler may over provision a host based on RAM (i.e., the RAM allocated by virtual machine instances may exceed the physical RAM).

This filter can be configured to allow a fixed amount of RAM overcommitment by using the `ram_allocation_ratio` configuration option in `nova.conf`. The default setting is:

```
ram_allocation_ratio=1.5
```

With this setting, if there is 1GB of free RAM, the scheduler will allow instances up to size 1.5GB to be run on that instance.

## RetryFilter

Filter out hosts that have already been attempted for scheduling purposes. If the scheduler selects a host to respond to a service request, and the host fails to respond to the request, this filter will prevent the scheduler from retrying that host for the service request.

This filter is only useful if the `scheduler_max_attempts` configuration option is set to a value greater than zero.

## SameHostFilter

Schedule the instance on the same host as another instance in a set of instances. To take advantage of this filter, the requester must pass a scheduler hint, using `same_host` as the key and a list of instance uuids as the value. This filter is the opposite of the `DifferentHostFilter`. Using the `nova` command-line tool, use the `--hint` flag:

```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 --  
hint same_host=a0cf03a5-d921-4877-bb5c-86d26cf818e1 --hint same_host=  
8c19174f-4220-44f0-824a-cd1eeef10287 server-1
```

With the API, use the `os:scheduler_hints` key:

```
{
    'server': {
        'name': 'server-1',
        'imageRef': 'cedef40a-ed67-4d10-800e-17455edce175',
        'flavorRef': '1'
    },
    'os:scheduler_hints': {
        'same_host': ['a0cf03a5-d921-4877-bb5c-86d26cf818e1',
                     '8c19174f-4220-44f0-824a-cd1eeef10287'],
    }
}
```

## SimpleCIDRAffinityFilter

Schedule the instance based on host IP subnet range. To take advantage of this filter, the requester must specify a range of valid IP address in CIDR format, by passing two scheduler hints:

`build_near_host_ip` The first IP address in the subnet (e.g., 192.168.1.1)

`cidr` The CIDR that corresponds to the subnet (e.g., /24)

Using the **nova** command-line tool, use the **--hint** flag. For example, to specify the IP subnet 192.168.1.1/24

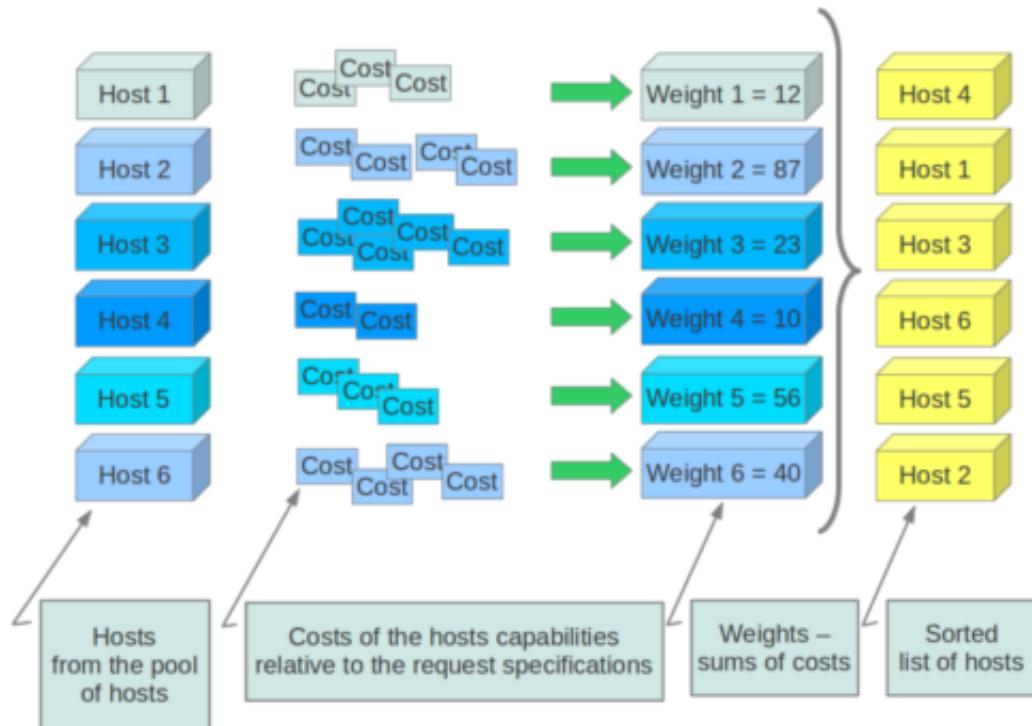
```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 --hint build_near_host_ip=192.168.1.1 --hint cidr=/24 server-1
```

With the API, use the `os:scheduler_hints` key:

```
{
    'server': {
        'name': 'server-1',
        'imageRef': 'cedef40a-ed67-4d10-800e-17455edce175',
        'flavorRef': '1'
    },
    'os:scheduler_hints': {
        'build_near_host_ip': '192.168.1.1',
        'cidr': '24'
    }
}
```

## Costs and Weights

Figure 12.2. Computing weighted costs



The Filter Scheduler takes the hosts that remain after the filters have been applied and applies one or more cost function to each host to get numerical scores for each host. Each cost score is multiplied by a weighting constant specified in the `nova.conf` config

file. The weighting constant configuration option is the name of the cost function, with the `_weight` string appended. Here is an example of specifying a cost function and its corresponding weight:

```
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn
compute_fill_first_cost_fn_weight=-1.0
```

Multiple cost functions can be specified in the `least_cost_functions` configuration option, separated by commas. For example:

```
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn,
nova.scheduler.least_cost.noop_cost_fn
compute_fill_first_cost_fn_weight=-1.0
noop_cost_fn_weight=1.0
```

If there are multiple cost functions, then the weighted cost scores are added together. The scheduler selects the host that has the minimum weighted cost.

The Compute service comes with three cost functions:

## **nova.scheduler.least\_cost.compute\_fill\_first\_cost\_fn**

This cost function calculates the amount of free memory (RAM) available on the node. Because the scheduler minimizes cost, if this cost function is used as a weight of +1, by doing:

```
compute_fill_first_cost_fn_weight=1.0
```

then the scheduler will tend to "fill up" hosts, scheduling virtual machine instances to the same host until there is no longer sufficient RAM to service the request, and then moving to the next node

If the user specifies a weight of -1 by doing:

```
compute_fill_first_cost_fn_weight=-1.0
```

then the scheduler will favor hosts that have the most amount of available RAM, leading to a "spread-first" behavior.

## **nova.scheduler.least\_cost.retry\_host\_cost\_fn**

This cost function adds additional cost for retrying scheduling a host that was already used for a previous scheduling attempt.

The normal method of using this function is to set `retry_host_cost_fn_weight` to a positive value, so that hosts which consistently encounter build failures will be used less often.

## **nova.scheduler.least\_cost.noop\_cost\_fn**

This cost function returns 1 for all hosts. It is a "no-op" cost function (i.e., it does not do anything to discriminate among hosts). In practice, this cost function is never used.

# Other Schedulers

While an administrator is likely to only need to work with the Filter Scheduler, Compute comes with other schedulers as well, described below.

## Chance Scheduler

The Chance Scheduler (`nova.scheduler.chance.ChanceScheduler`) randomly selects from the lists of filtered hosts. It is the default volume scheduler.

## Multi Scheduler

The Multi Scheduler `nova.scheduler.multi.MultiScheduler` holds multiple sub-schedulers, one for `nova-compute` requests and one for `cinder-volume` requests. It is the default top-level scheduler as specified by the `scheduler_driver` configuration option.

# Host aggregates

## Overview

Host aggregates are a mechanism to further partition an availability zone; while availability zones are visible to users, host aggregates are only visible to administrators. Host aggregates started out as a way to use Xen hypervisor resource pools, but has been generalized to provide a mechanism to allow administrators to assign key-value pairs to groups of machines. Each node can have multiple aggregates, each aggregate can have multiple key-value pairs, and the same key-value pair can be assigned to multiple aggregate. This information can be used in the scheduler to enable advanced scheduling, to set up Xen hypervisor resources pools or to define logical groups for migration.

## Command-line interface

The `nova` command-line tool supports the following aggregate-related commands.

<code>nova aggregate-list</code>	Print a list of all aggregates.
<code>nova aggregate-create &lt;name&gt; &lt;availability-zone&gt;</code>	Create a new aggregate named <code>&lt;name&gt;</code> in availability zone <code>&lt;availability-zone&gt;</code> . Returns the ID of the newly created aggregate.
<code>nova aggregate-delete &lt;id&gt;</code>	Delete an aggregate with id <code>&lt;id&gt;</code> .
<code>nova aggregate-details &lt;id&gt;</code>	Show details of the aggregate with id <code>&lt;id&gt;</code> .
<code>nova aggregate-add-host &lt;id&gt; &lt;host&gt;</code>	Add host with name <code>&lt;host&gt;</code> to aggregate with id <code>&lt;id&gt;</code> .
<code>nova aggregate-remove-host &lt;id&gt; &lt;host&gt;</code>	Remove the host with name <code>&lt;host&gt;</code> from the aggregate with id <code>&lt;id&gt;</code> .

<b>nova aggregate-set-metadata</b> <code>&lt;id&gt; &lt;key=value&gt;</code> <code>[&lt;key=value&gt; ...]</code>	Add or update metadata (key-value pairs) associated with the aggregate with id <code>&lt;id&gt;</code> .
<b>nova aggregate-update</b> <code>&lt;id&gt; &lt;name&gt;</code> <code>[&lt;availability_zone&gt;]</code>	Update the aggregate's name and optionally availability zone.
<b>nova host-list</b>	List all hosts by service.
<b>nova host-update – maintenance [enable   disable]</b>	Put/resume host into/from maintenance.



### Note

These commands are only accessible to administrators. If the username and tenant you are using to access the Compute service do not have the admin role, or have not been explicitly granted the appropriate privileges, you will see one of the following errors when trying to use these commands:

```
ERROR: Policy doesn't allow compute_extension:aggregates to be performed. (HTTP 403) (Request-ID: req-299fbff6-6729-4cef-93b2-e7e1f96b4864)
```

```
ERROR: Policy doesn't allow compute_extension:hosts to be performed. (HTTP 403) (Request-ID: req-ef2400f6-6776-4ea3-b6f1-7704085c27d1)
```

## Configure scheduler to support host aggregates

One common use case for host aggregates is when you want to support scheduling instances to a subset of compute hosts because they have a specific capability. For example, you may want to allow users to request compute hosts that have SSD drives if they need access to faster disk I/O, or access to compute hosts that have GPU cards to take advantage of GPU-accelerated code.

To configure the scheduler to support host aggregates, the `scheduler_default_filters` configuration option must contain the `AggregateInstanceExtraSpecsFilter` in addition to the other filters used by the scheduler. Add the following line to `/etc/nova/nova.conf` on the host that runs the `nova-scheduler` service to enable host aggregates filtering, as well as the other filters that are typically enabled:

```
scheduler_default_filters=AggregateInstanceExtraSpecsFilter,  
AvailabilityZoneFilter,RamFilter,ComputeFilter
```

## Example: specify compute hosts with SSDs

In this example, we configure the Compute service to allow users to request nodes that have solid-state drives (SSDs). We create a new host aggregate called `fast-io` in the availability zone called `nova`, we add the key-value pair `ssd=true` to the aggregate, and then we add compute nodes `node1`, and `node2` to it.

```
$ nova aggregate-create fast-io nova  
+-----+-----+-----+-----+-----+
```

```
| Id | Name      | Availability Zone | Hosts | Metadata |
+---+-----+-----+-----+-----+
| 1 | fast-io  | nova           |       |       |
+---+-----+-----+-----+
$ nova aggregate-set-metadata 1 ssd=true
+-----+-----+-----+-----+
| Id | Name      | Availability Zone | Hosts | Metadata      |
+-----+-----+-----+-----+
| 1 | fast-io  | nova           | []    | {u'ssd': u'true'} |
+-----+-----+-----+-----+
$ nova aggregate-add-host 1 node1
+-----+-----+-----+-----+
| Id | Name      | Availability Zone | Hosts      | Metadata      |
+-----+-----+-----+-----+
| 1 | fast-io  | nova           | [u'node1'] | {u'ssd': u'true'} |
+-----+-----+-----+-----+
$ nova aggregate-add-host 1 node2
+-----+-----+-----+-----+
| Id | Name      | Availability Zone | Hosts      | Metadata      |
|   |           |                   |           |               |
+-----+-----+-----+-----+
| 1 | fast-io  | nova           | [u'node1', u'node2'] | {u'ssd': u'true'} |
|   |           |                   |           |               |
+-----+-----+-----+-----+
|
```

Next, we use the **nova flavor-create** command to create a new flavor called `ssd.large` with an ID of 6, 8GB of RAM, 80GB root disk, and 4 vCPUs.

```
$ nova flavor-create ssd.large 6 8192 80 4
+-----+-----+-----+-----+-----+-----+-----+
+-----+-----+
| ID | Name      | Memory_MB | Disk | Ephemeral | Swap | VCPUs | RXTX_Factor |
| Is_Public | extra_specs |
+-----+-----+-----+-----+-----+-----+-----+
| 6 | ssd.large | 8192     | 80   | 0         |      | 4     | 1           |
| True | {}      |           |       |           |      |       |           |
+-----+-----+-----+-----+-----+-----+-----+
```

Once the flavor has been created, we specify one or more key-value pair that must match the key-value pairs on the host aggregates. In this case, there's only one key-value pair, `ssd=true`. Setting a key-value pair on a flavor is done using the **nova-manage instance\_type set\_key** command.

```
# nova-manage instance_type set_key --name=ssd.large --key=ssd --value=true
```

Once it is set, you should see the `extra_specs` property of the `ssd.large` flavor populated with a key of `ssd` and a corresponding value of `true`.

```
$ nova flavor-show ssd.large
+-----+-----+
| Property          | Value      |
+-----+-----+
| OS-FLV-DISABLED:disabled | False      |
| OS-FLV-EXT-DATA:ephemeral | 0          |
| disk              | 80         |
| extra_specs       | {u'ssd': u'true'} |
```

id	6
name	ssd.large
os-flavor-access:is_public	True
ram	8192
rxtx_factor	1.0
swap	
vcpus	4

Now, when a user requests an instance with the `ssd.large` flavor, the scheduler will only consider hosts with the `ssd=true` key-value pair. In this example, that would only be `node1` and `node2`.

## XenServer hypervisor pools to support live migration

When using the XenAPI-based hypervisor, the Compute service uses host aggregates to manage XenServer Resource pools, which are used in supporting live migration. See [Configuring Migrations](#) for details on how to create these kinds of host aggregates to support live migration.

# 13. Cells

*Cells* functionality allows you to scale an OpenStack Compute cloud in a more distributed fashion without having to use complicated technologies like database and message queue clustering. It is intended to support very large deployments.

When this functionality is enabled, the hosts in an OpenStack Compute cloud are partitioned into groups called cells. Cells are configured as a tree. The top-level cell should have a host that runs a `nova-api` service, but no `nova-compute` services. Each child cell should run all of the typical `nova-*` services in a regular Compute cloud except for `nova-api`. You can think of a cells as a normal Compute deployment in that each cell has its own database server and message queue broker.

The `nova-cells` service handles communication between cells and selecting a cell for new instances. This service is required for every cell. Communication between cells is pluggable, with the only option currently implemented being communication via RPC.

Cells scheduling is separate from host scheduling. `nova-cells` first picks a cell (currently randomly, future releases will add filtering/weighing functionality and decisions can be based on broadcasts of capacity/capabilities). Once a cell has been selected and the new build request has reached its `nova-cells` service, it will be sent over to the host scheduler in that cell and the build proceeds as it does without cells.



## Warning

Cell functionality is currently considered experimental.

## Cell configuration options

Cells are disabled by default. All cell-related configuration options go under a `[cells]` section in `nova.conf`. The following cell-related options are currently supported:

<code>enable</code>	Set this to True to turn on cell functionality, which is off by default.
<code>name</code>	Name of the current cell. This must be unique for each cell.
<code>capabilities</code>	List of arbitrary <code>key=value</code> pairs defining capabilities of the current cell. These are sent to parent cells, but aren't used in scheduling until later filter/weight support is added.
<code>call_timeout</code>	How long to wait for replies from calls between cells.

## Configuring the API (top-level) cell

The compute API class must be changed in the API cell so that requests can be proxied via `nova-cells` down to the correct cell properly. Add the following to `nova.conf` in the API cell:

```
[DEFAULT]
compute_api_class=nova.compute.cells_api.ComputeCellsAPI
...
[cells]
enable=True
name=api
```

## Configuring the child cells

Add the following to `nova.conf` in the child cells, replacing `cell1` with the name of each cell:

```
[DEFAULT]
# Disable quota checking in child cells. Let API cell do it exclusively.
quota_driver=nova.quota.NoopQuotaDriver

[cells]
enable=True
name=cell1
```

## Configuring the database in each cell

Before bringing the services online, the database in each cell needs to be configured with information about related cells. In particular, the API cell needs to know about its immediate children, and the child cells need to know about their immediate agents. The information needed is the RabbitMQ server credentials for the particular cell.

Use the `nova-manage cell create` command to add this information to the database in each cell:

```
$ nova-manage cell create -h
Options:
-h, --help                  show this help message and exit
--name=<name>              Name for the new cell
--cell_type=<parent|child>
                           Whether the cell is a parent or child
--username=<username>
                           Username for the message broker in this cell
--password=<password>
                           Password for the message broker in this cell
--hostname=<hostname>
                           Address of the message broker in this cell
--port=<number>
                           Port number of the message broker in this cell
--virtual_host=<virtual_host>
                           The virtual host of the message broker in this cell
--woffset=<float>
--wscale=<float>
```

As an example, assume we have an API cell named `api` and a child cell named `cell1`. Within the `api` cell, we have the following RabbitMQ server info:

```
rabbit_host=10.0.0.10
rabbit_port=5672
rabbit_username=api_user
rabbit_password=api_passwd
rabbit_virtual_host=api_vhost
```

And in the child cell named `cell1` we have the following RabbitMQ server info:

```
rabbit_host=10.0.1.10
rabbit_port=5673
rabbit_username=cell1_user
rabbit_password=cell1_passwd
rabbit_virtual_host=cell1_vhost
```

We would run this in the API cell, as root.

```
# nova-manage cell create --name=cell1 --cell_type=child --username=cell1_user
--password=cell1_passwd --hostname=10.0.1.10 --port=5673 --virtual_host=
cell1_vhost --woffset=1.0 --wscale=1.0
```

Repeat the above for all child cells.

In the child cell, we would run the following, as root:

```
# nova-manage cell create --name=api --cell_type=parent --username=api1_user
--password=api1_passwd --hostname=10.0.0.10 --port=5672 --virtual_host=
api_vhost --woffset=1.0 --wscale=1.0
```

# 14. System Administration

## Table of Contents

Understanding the Compute Service Architecture .....	267
Managing Compute Users .....	268
Managing the Cloud .....	268
Usage statistics .....	270
Managing logs .....	272
Reference Information for Securing with Root Wrappers .....	273
Using Migration .....	274
Recovering from a failed compute node .....	276
Recovering from a UID/GID mismatch .....	278
Nova Disaster Recovery Process .....	278

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

The OpenStack Compute cloud works via the interaction of a series of daemon processes named nova-\* that reside persistently on the host machine or machines. These binaries can all run on the same machine or be spread out on multiple boxes in a large deployment. The responsibilities of Services, Managers, and Drivers, can be a bit confusing at first. Here is an outline the division of responsibilities to make understanding the system a little bit easier.

Currently, Services are nova-api, nova-objectstore (which can be replaced with Glance, the OpenStack Image Service), nova-compute, and nova-network. Managers and Drivers are specified by configuration options and loaded using `utils.load_object()`. Managers are responsible for a certain aspect of the system. It is a logical grouping of code relating to a portion of the system. In general other components should be using the manager to make changes to the components that it is responsible for.

- nova-api - The nova-api service receives xml requests and sends them to the rest of the system. It is a wsgi app that routes and authenticate requests. It supports the EC2 and OpenStack APIs. There is a `nova-api.conf` file created when you install Compute.
- nova-objectstore - The nova-objectstore service is an ultra simple file-based storage system for images that replicates most of the S3 API. It can be replaced with OpenStack Image Service and a simple image manager or use OpenStack Object Storage as the virtual machine image storage facility. It must reside on the same node as `nova-compute`.
- nova-compute - The `nova-compute` service is responsible for managing virtual machines. It loads a Service object which exposes the public methods on `ComputeManager` via Remote Procedure Call (RPC).
- nova-network - The `nova-network` service is responsible for managing floating and fixed IPs, DHCP, bridging and VLANs. It loads a Service object which exposes the public

---

methods on one of the subclasses of NetworkManager. Different networking strategies are available to the service by changing the network\_manager configuration option to FlatManager, FlatDHCPManager, or VlanManager (default is VLAN if no other is specified).

## Understanding the Compute Service Architecture

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

### API Server

At the heart of the cloud framework is an API Server. This API Server makes command and control of the hypervisor, storage, and networking programmatically available to users in realization of the definition of cloud computing.

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

### Message Queue

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

A typical message passing event begins with the API server receiving a request from a user. The API server authenticates the user and ensures that the user is permitted to issue the subject command. 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 hostname. When such listening produces a work request, the worker takes assignment of the task and begins its execution. Upon completion, a response is dispatched to the queue which is received by the API server and relayed to the originating user. Database entries are queried, added, or removed as necessary throughout the process.

### Compute Worker

Compute workers manage computing instances on host machines. Through the API, commands are dispatched to compute workers to:

- 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

## Managing Compute Users

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

In order to begin using nova, you will need to create a user with the Identity Service.

## Managing the Cloud

There are three main tools that a system administrator will find useful to manage their cloud; the nova client, the nova-manage command, and the Euca2ools commands.

The nova-manage command may only be run by cloud administrators. Both novaclient and euca2ools can be used by all users, though specific commands may be restricted by Role Based Access Control in the Identity Management service.

## Using the nova command-line tool

Installing the python-novaclient gives you a `nova` shell command that enables Compute API interactions from the command line. You install the client, and then provide your username and password, set as environment variables for convenience, and then you can have the ability to send commands to your cloud on the command-line.

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

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

```
$ tar -zxvf python-novaclient-2.6.3.tar.gz
$ cd python-novaclient-2.6.3
$ sudo python setup.py install
```

Now that you have installed the python-novaclient, confirm the installation by entering:

```
$ nova help
```

```
usage: nova [--debug] [--os-username OS_USERNAME] [--os-password OS_PASSWORD]
            [--os-tenant-name_name OS_TENANT_NAME] [--os-auth-url OS_AUTH_URL]
            [--os-region-name OS_REGION_NAME] [--service-type SERVICE_TYPE]
            [--service-name SERVICE_NAME] [--endpoint-type ENDPOINT_TYPE]
            [--version VERSION]
            <subcommand> ...
```

In return, you will get a listing of all the commands and parameters for the nova command line client. By setting up the required parameters as environment variables, you can fly through these commands on the command line. You can add --os-username on the nova command, or set them as environment variables:

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

Using the Identity Service, you are supplied with an authentication endpoint, which nova recognizes as the OS\_AUTH\_URL.

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

## Using the nova-manage command

The nova-manage command may be used to perform many essential functions for administration and ongoing maintenance of nova, such as network creation or user manipulation.

The man page for nova-manage has a good explanation for each of its functions, and is recommended reading for those starting out. Access it by running:

```
$ man nova-manage
```

For administrators, the standard pattern for executing a nova-manage command is:

```
$ nova-manage category command [args]
```

For example, to obtain a list of all projects: **nova-manage project list**

Run without arguments to see a list of available command categories: **nova-manage**

You can also run with a category argument such as user to see a list of all commands in that category: nova-manage service

## Using the euca2ools commands

For a command-line interface to EC2 API calls, use the euca2ools command line tool. It is documented at [http://open.eucalyptus.com/wiki/Euca2oolsGuide\\_v1.3](http://open.eucalyptus.com/wiki/Euca2oolsGuide_v1.3)

## Usage statistics

The nova command-line tool can provide some basic statistics on resource usage for hosts and instances.

For more sophisticated monitoring, see the [Ceilometer](#) project, which is currently under development. You may also wish to consider installing tools such as [Ganglia](#) or [Graphite](#) if you require access to more detailed data.

## Host usage statistics

Use the **nova host-list** command to list the hosts and the nova-related services that are running on them:

```
$ nova host-list
+-----+-----+
| host_name | service |
+-----+-----+
| c2-compute-01 | compute |
| c2-compute-01 | network |
| c2-compute-02 | compute |
| c2-compute-02 | network |
| c2-compute-03 | compute |
| c2-compute-03 | network |
| c2-compute-04 | compute |
| c2-compute-04 | network |
| c2-controller-01 | cert |
| c2-controller-01 | consoleauth |
| c2-controller-01 | scheduler |
+-----+-----+
```

Use the **nova host-describe** command to retrieve a summary of resource usage of all of the instances running on the host. The "cpu" column is the sum of the virtual CPUs of all of the instances running on the host, the "memory\_mb" column is the sum of the memory (in MB) allocated to the instances running on the hosts, and the "disk\_gb" column is the sum of the root and ephemeral disk sizes (in GB) of the instances running on the hosts.

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

```
$ nova host-describe c2-compute-01
+-----+-----+-----+
+-----+-----+-----+
| HOST | PROJECT |      | cpu | memory_mb | disk_gb
|      |          |      |-----+-----+-----+
|      |          |      |-----+-----+-----+
```

	c2-compute-01	(total)		24	96677
	c2-compute-01	(used_max)		2	2560
	c2-compute-01	(used_now)		4	7168
	c2-compute-01	f34d8f7170034280a42f6318d1a4af34	2	2560	0

## Instance usage statistics

Use the **nova diagnostics** command to retrieve CPU, memory, I/O and network statistics from an instance:

\$ nova diagnostics ubuntu	
Property	Value
cpu0_time	1138410000000
memory	524288
memory-actual	524288
memory-rss	591664
vda_errors	-1
vda_read	334864384
vda_read_req	13851
vda_write	2985382912
vda_write_req	177180
vnet4_rx	45381339
vnet4_rx_drop	0
vnet4_rx_errors	0
vnet4_rx_packets	106426
vnet4_tx	37513574
vnet4_tx_drop	0
vnet4_tx_errors	0
vnet4_tx_packets	162200

Use the **nova usage-list** command to get summary statistics for each tenant:

\$ nova usage-list		Usage from 2012-10-10 to 2012-11-08:				
Tenant ID	Disk GB-Hours	Instances	RAM MB-Hours	CPU Hours		
0eec5c34a7a24a7a8ddad27cb81d2706   00	8	240031.10	468.81	0.		
92a5d9c313424537b78ae3e42858fd4e   00	5	483568.64	236.12	0.		
f34d8f7170034280a42f6318d1a4af34   00	106	16888511.58	9182.88	0.		

# Managing logs

## Logging module

Adding the following line to `/etc/nova/nova.conf` will allow you to specify a configuration file for changing the logging behavior, in particular for changing the logging level (e.g., DEBUG, INFO, WARNING, ERROR):

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

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

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

This example sets the debugging level to `INFO` (which less verbose than the default `DEBUG` setting). See the [Python documentation on logging configuration file format](#) for more details on this file, including the meaning of the `handlers` and `quaname` variables. See [etc/nova/logging\\_sample.conf](#) in the openstack/nova repository on GitHub for an example `logging.conf` file with various handlers defined.

## Syslog

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

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

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

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



### Note

While the example above uses the same local facility for each service (`LOG_LOCAL0`, which corresponds to syslog facility `LOCAL0`), we recommend

that you configure a separate local facility for each service, as this provides better isolation and more flexibility. For example, you may want to capture logging info at different severity levels for different services. Syslog allows you to define up to seven local facilities, LOCAL0, LOCAL1, ..., LOCAL7. See the syslog documentation for more details.

## Rsyslog

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

This example shows a minimal configuration for `/etc/rsyslog.conf` on the log server host which will receive the log files:

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

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

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

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

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

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

## Reference Information for Securing with Root Wrappers

The goal of the root wrapper is to allow the nova unprivileged user to run a number of actions as the root user, in the safest manner possible. Historically, Nova used a specific sudoers file listing every command that the nova user was allowed to run, and just used sudo to run that command as root. However this was difficult to maintain (the sudoers file was in packaging), and did not allow for complex filtering of parameters (advanced filters). The rootwrap was designed to solve those issues.

How rootwrap works:

Instead of just calling sudo make me a sandwich, Compute services starting with nova- call sudo nova-rootwrap /etc/nova/rootwrap.conf make me a sandwich. A generic sudoers entry lets the nova user run nova-rootwrap as root. The nova-rootwrap code looks for filter definition directories in its configuration file, and loads command filters from them. Then it

checks if the command requested by Compute matches one of those filters, in which case it executes the command (as root). If no filter matches, it denies the request.

## Security model

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

## Details of rootwrap.conf

The `rootwrap.conf` file is used to influence how nova-rootwrap works. Since it's in the trusted security path, it needs to be owned and writeable only by the root user. Its location is specified both in the sudoers entry and in the `nova.conf` configuration file with the `rootwrap_config`= entry.

It uses an INI file format with the following sections and parameters:

**Table 14.1. Description of rootwrap.conf configuration options**

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

## Details of .filters files

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

It uses an INI file format with a [Filters] section and several lines, each with a unique parameter name (different for each filter you define):

**Table 14.2. Description of rootwrap.conf configuration options**

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

## Using Migration

Before starting migrations, review the [Configuring Migrations](#) section.

Migration provides a scheme to migrate running instances from one OpenStack Compute server to another OpenStack Compute server. This feature can be used as described below.

- First, look at the running instances, to get the ID of the instance you wish to migrate.

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

Second, look at information associated with that instance - our example is vm1 from above.

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

In this example, vm1 is running on HostB.

- Third, select the server to migrate instances to.

```
# nova-manage service list
HostA nova-scheduler enabled :-) None
HostA nova-network enabled :-) None
HostB nova-compute enabled :-) None
HostC nova-compute enabled :-) None
```

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

- Third, ensure that HostC has enough resource for migration.

```
# nova-manage service describe_resource HostC
HOST          PROJECT      cpu    mem(mb)      hdd
HostC(total)        16    32232       878
HostC(used_now)     13    21284       442
HostC(used_max)     13    21284       442
HostC            p1        5    10240       150
HostC            p2        5    10240       150
....
```

- **cpu:**the number of cpu
- **mem(mb):**total amount of memory (MB)
- **hdd:**total amount of space for NOVA-INST-DIR/instances(GB)
- **1st line shows** total amount of resource physical server has.
- **2nd line shows** current used resource.
- **3rd line shows** maximum used resource.
- **4th line and under** is used resource per project.
- Finally, use the **nova live-migration** command to migrate the instances.

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

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



### Note

While the nova command is called **live-migration**, under the default Compute configuration options the instances are suspended before migration. See the [Configuring Migrations](#) section for more details.

## Recovering from a failed compute node

If you have deployed OpenStack Compute with a shared filesystem, you can quickly recover from a failed compute node.

## Using the evacuate API for KVM/libvirt

Refer to [Evacuate API Reference](#) section for more details.

## Manual recovery

For KVM/libvirt compute node recovery refer to section above, while the guide below may be applicable for other hypervisors.

## Working with host information

The first step is to identify the vms on the affected hosts, using tools such as a combination of `nova list` and `nova show` or `euca-describe-instances`. Here's an example using the EC2 API - instance i-000015b9 that is running on node np-rcc54:

```
i-000015b9 at3-ui02 running nectarkey (376, np-rcc54) 0 m1.xxlarge
2012-06-19T00:48:11.000Z 115.146.93.60
```

First, you can review the status of the host using the nova database, some of the important information is highlighted below. This example converts an EC2 API instance ID into an openstack ID - if you used the `nova commands`, you can substitute the ID directly. You can find the credentials for your database in `/etc/nova.conf`.

```
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
...
```

## Recover the VM

Armed with the information of VMs on the failed host, determine which compute host the affected VMs should be moved to. In this case, the VM will move to np-rcc46, which is achieved using this database command:

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

Next, if using a hypervisor that relies on libvirt (such as KVM) it is a good idea to update the `libvirt.xml` file (found in `/var/lib/nova/instances/[instance ID]`). The important changes to make are to change the `DHCPSERVER` value to the host ip address of the nova compute host that is the VMs new home, and update the VNC IP if it isn't already 0.0.0.0.

Next, reboot the VM:

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

In theory, the above database update and `nova reboot` command are all that is required to recover the VMs from a failed host. However, if further problems occur, consider looking at recreating the network filter configuration using `virsh`, restarting the nova services or updating the `vm_state` and `power_state` in the nova database.

## Recovering from a UID/GID mismatch

When running OpenStack compute, using a shared filesystem or an automated configuration tool, you could encounter a situation where some files on your compute node are using the wrong UID or GID. This causes a raft of errors, such as being unable to live migrate, or start virtual machines.

The following is a basic procedure run on nova-compute hosts, based on the KVM hypervisor, that could help to restore the situation:

- First, make sure you don't use numbers that are already used for some other user/group.
- Set the nova uid in `/etc/passwd` to the same number in all hosts (e.g. 112)
- Set the libvirt-qemu uid in `/etc/passwd` to the same number in all hosts (e.g. 119)
- Set the nova group in `/etc/group` file to the same number in all hosts (e.g. 120)
- Set the libvirdt group in `/etc/group` file to the same number in all hosts (e.g. 119)
- Stop the services on the compute node
- Change all the files owned by nova or group by nova, e.g.

```
find / -uid 108 -exec chown nova {} \; # note the 108 here is the
old nova uid before the change
find / -gid 120 -exec chgrp nova {} \;
```

- Repeat the steps for the libvirt-qemu owned files if those were needed to change
- Restart the services

Following this, you can run the `find` command to verify that all files using the correct identifiers.

## Nova Disaster Recovery Process

Sometimes, things just don't go right. An incident is never planned, by its definition.

In this section, we will review managing your cloud after a disaster, and how to easily backup the persistent storage volumes, which is another approach when you face a disaster. Even apart from the disaster scenario, backup ARE mandatory.

For reference, you can find a DRP definition here : [http://en.wikipedia.org/wiki/Disaster\\_Recovery\\_Plan](http://en.wikipedia.org/wiki/Disaster_Recovery_Plan).

## A- The disaster Recovery Process presentation

A disaster could happen to several components of your architecture : a disk crash, a network loss, a power cut, etc. In this example, we suppose the following setup :

1. A cloud controller (nova-api, nova-objectstore, nova-network)
2. A compute node (nova-compute)
3. A Storage Area Network used by cinder-volumes (aka SAN)

The example disaster will be the worst one : a power loss. That power loss applies to the three components. *Let's see what runs and how it runs before the crash :*

- From the SAN to the cloud controller, we have an active iscsi session (used for the "cinder-volumes" LVM's VG).
- From the cloud controller to the compute node we also have active iscsi sessions (managed by cinder-volume).
- For every volume an iscsi session is made (so 14 ebs volumes equals 14 sessions).
- From the cloud controller to the compute node, we also have iptables/ ebtables rules which allows the access from the cloud controller to the running instance.
- And at least, from the cloud controller to the compute node ; saved into database, the current state of the instances (in that case "running" ), and their volumes attachment (mountpoint, volume id, volume status, etc..)

Now, after the power loss occurs and all hardware components restart, the situation is as follows:

- From the SAN to the cloud, the ISCSI session no longer exists.
- From the cloud controller to the compute node, the ISCSI sessions no longer exist.
- From the cloud controller to the compute node, the iptables and ebtables are recreated, since, at boot, nova-network reapply the configurations.
- From the cloud controller, instances turn into a shutdown state (because they are no longer running)
- Into the database, data was not updated at all, since nova could not have guessed the crash.

Before going further, and in order to prevent the admin to make fatal mistakes, **the instances won't be lost**, since no "destroy" or "terminate" command had been invoked, so the files for the instances remain on the compute node.

The plan is to perform the following tasks, in that exact order. Any extra step would be dangerous at this stage :

1. We need to get the current relation from a volume to its instance, since we will recreate the attachment.

2. We need to update the database in order to clean the stalled state. (After that, we won't be able to perform the first step).
3. We need to restart the instances (so go from a "shutdown" to a "running" state).
4. After the restart, we can reattach the volumes to their respective instances.
5. That step, which is not a mandatory one, exists in an SSH into the instances in order to reboot them.

## B - The Disaster Recovery Process itself

- **Instance to Volume relation**

We need to get the current relation from a volume to its instance, since we will recreate the attachment :

This relation could be figured by running **nova volume-list** (note that nova client includes ability to get volume info from cinder)

- **Database Update**

Second, we need to update the database in order to clean the stalled state. Now that we have saved the attachments we need to restore for every volume, the database can be cleaned with the following 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;
```

Now, when running **nova volume-list** all volumes should be available.

- **Instances Restart**

We need to restart the instances. This can be done via a simple **nova reboot \$instance**

At that stage, depending on your image, some instances will completely reboot and become reachable, while others will stop on the "plymouth" stage.

**DO NOT reboot a second time** the ones which are stopped at that stage (see below, the fourth step). In fact it depends on whether you added an /etc/fstab entry for that volume or not. Images built with the *cloud-init* package will remain on a pending state, while others will skip the missing volume and start. (More information is available on [help.ubuntu.com](http://help.ubuntu.com)) But remember that the idea of that stage is only to ask nova to reboot every instance, so the stored state is preserved.

- **Volume Attachment**

After the restart, we can reattach the volumes to their respective instances. Now that nova has restored the right status, it is time to perform the attachments via a **nova volume-attach**

Here is a simple snippet that uses the file we created :

```
#!/bin/bash

while read line; do
    volume=`echo $line | $CUT -f 1 -d " "`
    instance=`echo $line | $CUT -f 2 -d " "`
    mount_point=`echo $line | $CUT -f 3 -d " "`
    echo "ATTACHING VOLUME FOR INSTANCE - $instance"
    nova volume-attach $instance $volume $mount_point
    sleep 2
done < $volumes_tmp_file
```

At that stage, instances which were pending on the boot sequence (*plymouth*) will automatically continue their boot, and restart normally, while the ones which booted will see the volume.

- **SSH into instances**

If some services depend on the volume, or if a volume has an entry into fstab, it could be good to simply restart the instance. This restart needs to be made from the instance itself, not via nova. So, we SSH into the instance and perform a reboot :

```
$ shutdown -r now
```

Voila! You successfully recovered your cloud after that.

Here are some suggestions :

- Use the parameter `errors=remount` in the fstab file, which will prevent data corruption.

The system would lock any write to the disk if it detects an I/O error. This configuration option should be added into the cinder-volume server (the one which performs the ISCSI connection to the SAN), but also into the instances' fstab file.

- Do not add the entry for the SAN's disks to the cinder-volume's fstab file.

Some systems will hang on that step, which means you could lose access to your cloud-controller. In order to re-run the session manually, you would run the following command before performing the mount:

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

- For your instances, if you have the whole `/home/` directory on the disk, instead of emptying the `/home` directory and map the disk on it, leave a user's directory with the user's bash files and the `authorized_keys` file.

This will allow you to connect to the instance, even without the volume attached, if you allow only connections via public keys.

## C- Scripted DRP

You can download from [here](#) a bash script which performs these five steps :

The "test mode" allows you to perform that whole sequence for only one instance.

To reproduce the power loss, connect to the compute node which runs that same instance and close the iscsi session. Do not detach the volume via `nova volume-detach`, but instead manually close the iscsi session.

In the following example, the iscsi session is number 15 for that instance :

```
$ iscsidadm -m session -u -r 15
```

**Do not forget the flag `-r`; otherwise, you will close ALL sessions.**

# 15. OpenStack Interfaces

## Table of Contents

About the Dashboard .....	283
Remote Console Access .....	294

OpenStack has components that provide a view of the OpenStack installation such as a Django-built website that serves as a dashboard and the ability to connect to running instances using a VNC connection via a VNC Proxy.

## About the Dashboard

You can use a dashboard interface with an OpenStack Compute installation with a web-based console provided by the OpenStack Dashboard project. It provides web-based interactions with the OpenStack Compute cloud controller through the OpenStack APIs. These instructions are for an example deployment configured with an Apache web server.

## System Requirements for the Dashboard

Because Apache does not serve content from a root user, you must use another user with sudo privileges and run as that user.

You should have a running OpenStack Compute installation with the Identity Service, Keystone, enabled for identity management.

The dashboard needs to be installed on the node that can contact the Identity Service.

You should know the URL of your Identity endpoint and the Compute endpoint.

You must know the credentials of a valid Identity service user.

You must have git installed. It's straightforward to install it with sudo apt-get install git-core.

Python 2.6 is required, and these instructions have been tested with Ubuntu 10.10. It should run on any system with Python 2.6 or 2.7 that is capable of running Django including Mac OS X (installing prerequisites may differ depending on platform).

Optional components:

- An Image Store (*Glance*) endpoint.
- An Object Store (*Swift*) endpoint.
- A [Quantum](#) (networking) endpoint.

## Installing the OpenStack Dashboard

Here are the overall steps for creating the OpenStack dashboard. Details about deployment are available at [Deploying Horizon](#).

1. Install the OpenStack Dashboard framework including Apache and related modules.
2. Configure the Dashboard.
3. Restart and run the Apache server.

Install the OpenStack Dashboard, as root:

```
# apt-get install -y memcached libapache2-mod-wsgi openstack-dashboard  
# yum install -y memcached python-memcached mod_wsgi openstack-dashboard
```

Next, modify the variable `CACHE_BACKEND` in `/etc/openstack-dashboard/local_settings.py` to match the ones set in `/etc/memcached.conf`/`/etc/sysconfig/memcached.conf`. Open `/etc/openstack-dashboard/local_settings.py`/`/etc/openstack-dashboard/local_settings` and look for this line:

```
CACHE_BACKEND = 'memcached://127.0.0.1:11211'
```



### Note

The address and port in the new value need to be equal to the ones set in `/etc/memcached.conf`/`/etc/sysconfig/memcached`.

If you change the memcached settings, restart the Apache web server for the changes to take effect.



### Note

This guide has selected memcache as a session store for OpenStack Dashboard. There are other options available, each with benefits and drawbacks. Refer to the OpenStack Dashboard Session Storage section for more information.



### Note

In order to change the timezone you can use either dashboard or inside `/etc/openstack-dashboard/local_settings`/`/etc/openstack-dashboard/local_settings.py` you can change below mentioned parameter.

```
TIME_ZONE = "UTC"
```

## Configuring the Dashboard

### 1. Simple deployment (HTTP)

Specify the host for your OpenStack Identity Service endpoint in the `/etc/openstack-dashboard/local_settings.py` file with the `OPENSTACK_HOST` setting. An example is included:

```
import os  
  
from django.utils.translation import ugettext_lazy as _  
  
DEBUG = False  
TEMPLATE_DEBUG = DEBUG
```

```
PROD = True
USE_SSL = False

SITE_BRANDING = 'OpenStack Dashboard'

# Ubuntu-specific: Enables an extra panel in the 'Settings' section
# that easily generates a Juju environments.yaml for download,
# preconfigured with endpoints and credentials required for bootstrap
# and service deployment.
ENABLE_JUJU_PANEL = True

# Note: You should change this value
SECRET_KEY = 'elj1IWl0WHgryYxFT6j7cM5fGO0xWY0'

# Specify a regular expression to validate user passwords.
# HORIZON_CONFIG = {
#     "password_validator": {
#         "regex": '.*',
#         "help_text": _("Your password does not meet the requirements."))
#     }
# }

LOCAL_PATH = os.path.dirname(os.path.abspath(__file__))

CACHE_BACKEND = 'memcached://127.0.0.1:11211/'

# Send email to the console by default
EMAIL_BACKEND = 'django.core.mail.backends.console.EmailBackend'
# Or send them to /dev/null
#EMAIL_BACKEND = 'django.core.mail.backends.dummy.EmailBackend'

# Configure these for your outgoing email host
# EMAIL_HOST = 'smtp.my-company.com'
# EMAIL_PORT = 25
# EMAIL_HOST_USER = 'djangomail'
# EMAIL_HOST_PASSWORD = 'top-secret!'

# For multiple regions uncomment this configuration, and add (endpoint,
# title).
# AVAILABLE_REGIONS = [
#     ('http://cluster1.example.com:5000/v2.0', 'cluster1'),
#     ('http://cluster2.example.com:5000/v2.0', 'cluster2'),
# ]

OPENSTACK_HOST = "127.0.0.1"
OPENSTACK_KEYSTONE_URL = "http://%s:5000/v2.0" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "Member"

# The OPENSTACK_KEYSTONE_BACKEND settings can be used to identify the
# capabilities of the auth backend for Keystone.
# If Keystone has been configured to use LDAP as the auth backend then set
# can_edit_user to False and name to 'ldap'.
#
# TODO(tres): Remove these once Keystone has an API to identify auth
# backend.
OPENSTACK_KEYSTONE_BACKEND = {
    'name': 'native',
    'can_edit_user': True
}
```

```
# OPENSTACK_ENDPOINT_TYPE specifies the endpoint type to use for the
# endpoints
# in the Keystone service catalog. Use this setting when Horizon is running
# external to the OpenStack environment. The default is 'internalURL'.
#OPENSTACK_ENDPOINT_TYPE = "publicURL"

# The number of Swift containers and objects to display on a single page
# before
# providing a paging element (a "more" link) to paginate results.
API_RESULT_LIMIT = 1000

# If you have external monitoring links, eg:
# EXTERNAL_MONITORING = [
#     ['Nagios','http://foo.com'],
#     ['Ganglia','http://bar.com'],
# ]

LOGGING = {
    'version': 1,
    # When set to True this will disable all logging except
    # for loggers specified in this configuration dictionary. Note that
    # if nothing is specified here and disable_existing_loggers is True,
    # django.db.backends will still log unless it is disabled
    explicitly.
    'disable_existing_loggers': False,
    'handlers': {
        'null': {
            'level': 'DEBUG',
            'class': 'django.utils.log.NullHandler',
        },
        'console': {
            # Set the level to "DEBUG" for verbose output logging.
            'level': 'INFO',
            'class': 'logging.StreamHandler',
        },
    },
    'loggers': {
        # Logging from django.db.backends is VERY verbose, send to null
        # by default.
        'django.db.backends': {
            'handlers': ['null'],
            'propagate': False,
        },
        'horizon': {
            'handlers': ['console'],
            'propagate': False,
        },
        'novaclient': {
            'handlers': ['console'],
            'propagate': False,
        },
        'keystoneclient': {
            'handlers': ['console'],
            'propagate': False,
        },
        'nose.plugins.manager': {
            'handlers': ['console'],
            'propagate': False,
        },
    }
}
```

```
}
```

The HORIZON\_CONFIG dictionary contains all the settings for the Dashboard. Whether or not a service is in the Dashboard depends on the Service Catalog configuration in the Identity service. Refer to [Horizon Settings and Configuration](#) for the full listing.

## 2. Secured deployment (HTTPS)

While the standard installation uses a non-encrypted channel (HTTP), it is possible to enable the SSL support for the OpenStack Dashboard. In the following example, we use the domain "http://openstack.example.com", make sure to use one that fits your current setup.

- In /etc/openstack-dashboard/local\_settings.py update the following directive:

```
USE_SSL = True
```

- Edit /etc/apache2/ports.conf and add the following line:

```
NameVirtualHost *:443
```

- Edit /etc/apache2/conf.d/openstack-dashboard.conf:

Before:

```
WSGIScriptAlias / /usr/share/openstack-dashboard/openstack_dashboard/wsgi/
django.wsgi
WSGIDaemonProcess horizon user=www-data group=www-data processes=3
threads=10
Alias /static /usr/share/openstack-dashboard/openstack_dashboard/static/
<Directory /usr/share/openstack-dashboard/openstack_dashboard/wsgi>
    Order allow,deny
    Allow from all
</Directory>
```

After:

```
<VirtualHost *:80>
    ServerName openstack.example.com
    RedirectPermanent / https://openstack.example.com
</VirtualHost>
<VirtualHost *:443>
    ServerName openstack.example.com

    SSLEngine On
    SSLCertificateFile /etc/apache2/SSL/openstack.example.com.crt
    SSLCACertificateFile /etc/apache2/SSL/openstack.example.com.crt
    SSLCertificateKeyFile /etc/apache2/SSL/openstack.example.com.key
    SetEnvIf User-Agent ".*MSIE.*" nokeepalive ssl-unclean-shutdown

    WSGIScriptAlias / /usr/share/openstack-dashboard/openstack_dashboard/
wsgi/django.wsgi
    WSGIDaemonProcess horizon user=www-data group=www-data processes=3
    threads=10
        Alias /static /usr/share/openstack-dashboard/openstack_dashboard/
static/
        <Directory /usr/share/openstack-dashboard/openstack_dashboard/wsgi>
```

```
Order allow,deny
Allow from all
</Directory>
</VirtualHost>
```

In this configuration, we instruct Apache to listen on the port 443 and to redirect all the hits to the HTTPs protocol for all the non-secured requests. In the secured section, we define as well the private key, the public key, and the certificate to use.

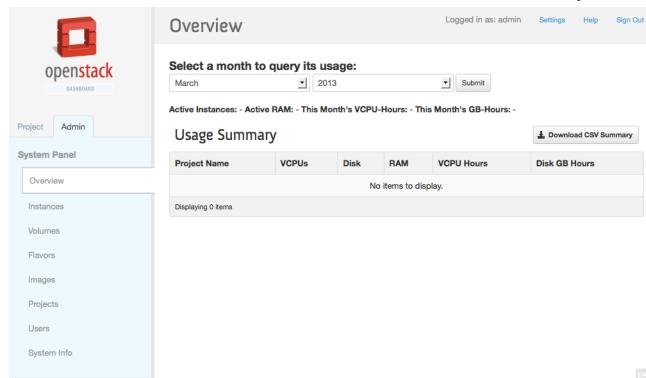
- Finally restart Apache and memcached:

```
# service apache2 restart
# service memcached restart
```

You should now be redirected to the HTTPs page if you call the HTTP version of the dashboard via your browser.

## Validating the Dashboard Install

To validate the Dashboard installation, point your browser at <http://192.168.206.130>. Once you connect to the Dashboard with the URL, you should see a login window. Enter the credentials for users you created with the Identity Service, Keystone. For example, enter "admin" for the username and "secrete" as the password.



## How To Custom Brand The OpenStack Dashboard (Horizon)

Adapted from a [blog post by Preston Lee](#).

When deploying OpenStack on [Ubuntu Server 12.04](#), you can have the `openstack-dashboard` package installed to provide the web-based "Horizon" GUI component. Canonical also provides an `openstack-dashboard-ubuntu-theme` package that brands the Python-based Django GUI.

The [Horizon documents](#) briefly mention branding customization to give you a head start, but here are more specific steps. Here's a custom-branded Horizon dashboard with custom colors, logo, and site title:

The screenshot shows the TGen Cloud login interface. At the top is a blue header bar with the text "TGen Cloud". Below it is a white "Log In" form. The form has two input fields: "User Name" containing "plee" and "Password" containing a series of asterisks. A "Sign In" button is at the bottom right of the form.

The screenshot shows the TGen Cloud dashboard with a dark blue header bar. On the left is a "System Panel" sidebar with links for Project, Admin, Overview, Instances, Services, **Flavors**, Images, Projects, Users, and Quotas. The main content area is titled "Flavors" and displays a table of flavor details. The table has columns for ID, Flavor Name, VCPUs, Memory, Root Disk, Ephemeral Disk, and Actions. It lists five flavors: m1.xlarge (ID 5), m1.large (ID 4), m1.medium (ID 3), m1.small (ID 2), and m1.tiny (ID 1). Each row includes a checkbox and a "Delete Flavor" button. The status bar at the bottom indicates "Displaying 5 items".

Once you know where to make the appropriate changes, it's super simple. Step-by-step:

1. Create a graphical logo with a transparent background. The text "TGen Cloud" in this example is actually rendered via .png files of multiple sizes created with a graphics program. Use a 200×27 for the logged-in banner graphic, and 365×50 for the login screen graphic.

2. Set the HTML title (shown at the top of the browser window) by adding the following line to `/etc/openstack-dashboard/local_settings.py`:  
`SITE_BRANDING = "Example, Inc. Cloud"`
3. Upload your new graphic files to:

```
/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/img/
```

4. Create a new CSS stylesheet – we'll call ours `custom.css` – in the directory:

```
/usr/share/openstack-dashboard/openstack_dashboard/static/dashboard/css/
```

5. Edit your CSS file using the following as a starting point for customization, which simply overrides the Ubuntu customizations made in the `ubuntu.css` file.

Change the colors and image file names as appropriate, though the relative directory paths should be the same.

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

6. Open the following HTML template in an editor:

```
/usr/share/openstack-dashboard/openstack_dashboard/templates/_stylesheets.
html
```

7. Add a line to include your new stylesheet pointing to `custom.css`: (I've highlighted the new line in **bold**.)

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

8. Restart apache just for good measure: `sudo service apache2 restart`  
`sudo service httpd restart`

9. Reload the dashboard in your browser and fine tune your CSS appropriate.

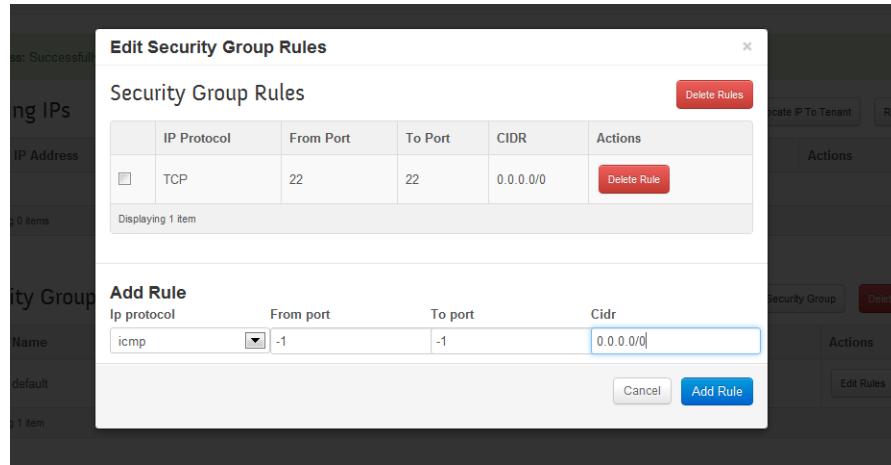
You're done!

## Launching Instances using Dashboard

The Dashboard can be used to launch instances. This section explains the various steps to be followed to launch a instance.

## Modify Security Groups

Before launching a VM, first modify the Security Groups rules to allow us to ping and SSH to the instances. This is done by editing the default security group or adding a new security group. For ease of understanding, modify the default security group.

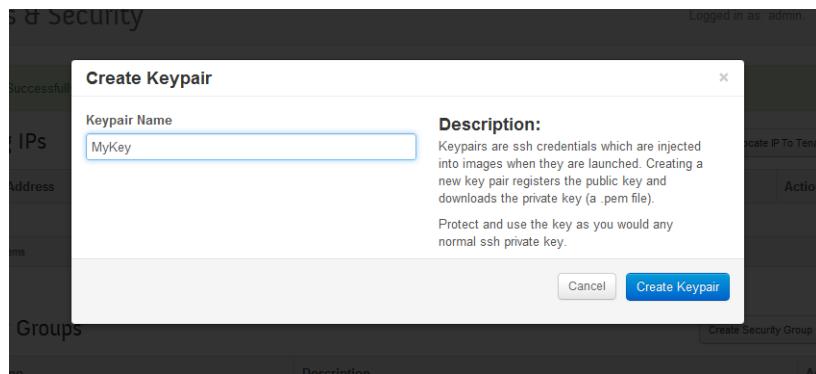


Select IP protocol TCP and enter 22 in "From Port" and "To Port" and CIDR 0.0.0.0/0. This opens port 22 for requests from any IP. If you want requests from particular range of IP, provide it in CIDR field.

Select IP protocol ICMP and enter -1 in "From Port" and "To Port" and CIDR 0.0.0.0/0. This allows ping from any IP. If you want ping requests from particular range of IP, provide it in CIDR field.

## Adding Keypair

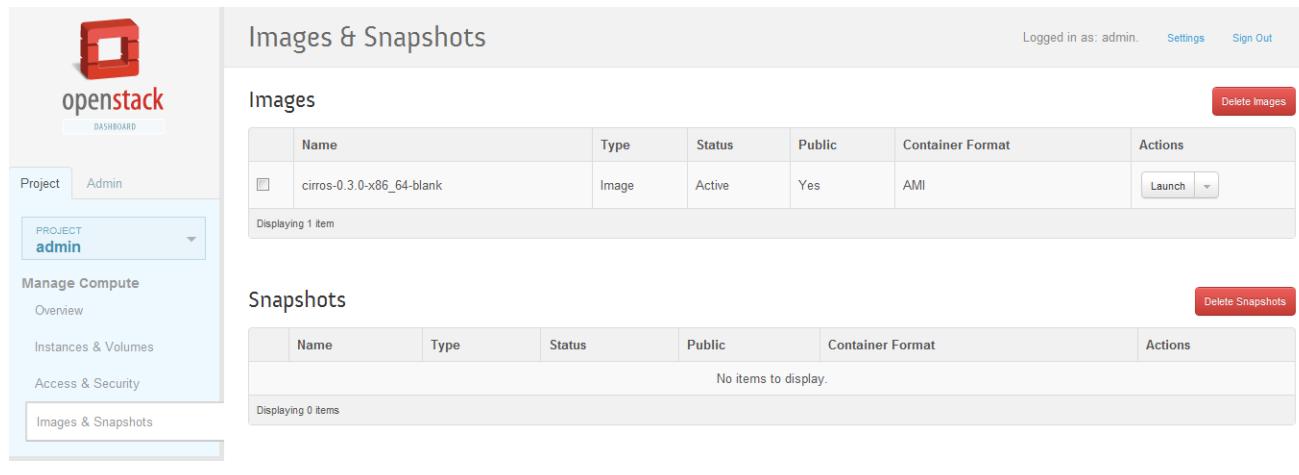
Next add a Keypair. Once a Keypair is added, the public key would be downloaded. This key can be used to SSH to the launched instance.



Once this is done, we are now all set to launch an Instance

## Launching Instance

Click Images & Snapshots and launch a required instance from the list of images available.



	Name	Type	Status	Public	Container Format	Actions
<input type="checkbox"/>	cirros-0.3.0-x86_64-blank	Image	Active	Yes	AMI	<button>Launch</button>

Displaying 1 item

	Name	Type	Status	Public	Container Format	Actions
No items to display.						

Displaying 0 items

Click launch on the required image. Provide a Server Name, select the flavor, the keypair added above and the default security group. Provide the number of instances required. Once these details are provided, click Launch Instance.

The 'Launch Instances' dialog box contains the following fields:

- Server Name: MyFirstInstance
- User Data: (Empty text area)
- Flavor: m1.tiny (1VCPU / 0GB Disk / 512MB Ram)
- Keypair: MyKey
- Instance Count: 1
- Security Groups: default (checkbox checked)

**Description:**

Specify the details for launching an instance. Also please make note of the table below, all tenants have quotas which define the limit of resources they are allowed to provision.

Quota Name	Limit
RAM (MB)	51200MB
Floating IPs	10
Instances	10
Volumes	10
Available Disk	1000GB

Once the status is Active, the instance is ready and we can ping and SSH to the instance.

The dashboard shows the following:

- Left Sidebar:** Project Admin, admin, Manage Compute Overview, Instances & Volumes, Access & Security, Images & Snapshots.
- Instances & Volumes Page:**
  - Instances:** A table with one row:

	Name	IP Address	Size	Status	Task	Power State	Actions
<input type="checkbox"/>	MyFirstInstance	10.0.0.2	512MB RAM   1 VCPU   0 Disk	Active	None	Running	<button>Edit Instance</button>

Displaying 1 item.
  - Volumes:** A table showing 'Displaying 0 items'.

## Make a secure connection to the launched instance

Here are the steps to SSH into an instance using the downloaded keypair file. The username is ubuntu for the Ubuntu cloud images on TryStack.

1. Download the MyKey.pem file from the OpenStack Dashboard.
2. In a command line interface, modify the access to the .pem file:

```
$ chmod 0600 MyKey.pem
```

3. Use the ssh-add command to ensure that the keypair is known to SSH:

```
$ ssh-add MyKey.pem
```

4. Copy the IP address from the MyFirstInstance.

5. Use the SSH command to make a secure connection to the instance:

```
$ ssh -i MyKey.pem ubuntu@10.0.0.2
```

You should see a prompt asking "Are you sure you want to continue connection (yes/no)?" Type yes and you have successfully connected.

## Remote Console Access

OpenStack has two main methods for providing a remote console or remote desktop access to guest Virtual Machines. They are VNC, and SPICE HTML5 and can be used either through the OpenStack dashboard and the command line. Best practice is to select one or the other to run.

### Overview of VNC Proxy

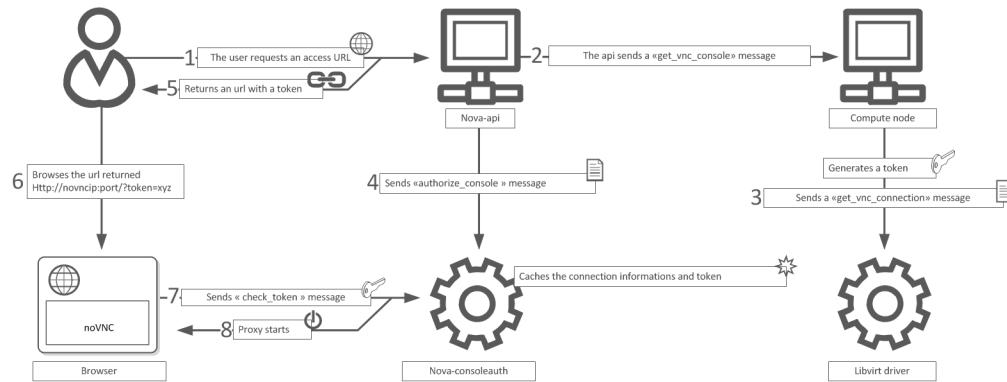
The VNC Proxy is an OpenStack component that allows users of the Compute service to access their instances through VNC clients.

The VNC console connection works as follows:

1. User connects to API and gets an access\_url like `http://ip:port/?token=xyz`.
2. User pastes URL in browser or as client parameter.
3. Browser/Client connects to proxy.
4. Proxy talks to nova-consoleauth to authorize the user's token, and then maps the token to the *private* host and port of an instance's VNC server. The compute host specifies the address the proxy should use to connect via the `nova.conf` option `vncserver_proxyclient_address`. In this way, the vnc proxy works as a bridge between the public network, and the private host network.
5. Proxy initiates connection to VNC server, and continues proxying until the session ends.

The proxy also performs the required function of tunneling the VNC protocol over Websockets so that the noVNC client has a way to talk VNC. Note that in general, the VNC proxy performs multiple functions:

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

**Figure 15.1. NoVNC Process**

## About nova-consoleauth

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

The `nova-consoleauth` shared service should not be confused with `nova-console`, which is a XenAPI-specific service that is not used by the most recent VNC proxy architecture.

## Typical Deployment

A typical deployment will consist of the following components:

- One `nova-consoleauth` process. Typically this runs on the controller host.
- One or more `nova-novncproxy` services. This supports browser-based novnc clients. For simple deployments, this service typically will run on the same machine as `nova-api`, since it proxies between the public network and the private compute host network.
- One or more `nova-xvpvncproxy` services. This supports the special Java client discussed in this document. For simple deployments, this service typically will run on the same machine as `nova-api`, since it proxies between the public network and the private compute host network.
- One or more compute hosts. These compute hosts must have correctly configured configuration options, as described below.

## Getting an Access URL

Nova provides the ability to create `access_urls` through the `os-consoles` extension. Support for accessing this URL is provided by `novclient`:

```
$ nova get-vnc-console [server_id] [novnc/xvpvnc]
```

Specify 'novnc' to retrieve a URL suitable for pasting into a web browser. Specify 'xvpvnc' for a URL suitable for pasting into the Java client.

So to request a web browser URL:

```
$ nova get-vnc-console [server_id] novnc
```

## VNC Configuration Options

**Table 15.1. Description of nova.conf file configuration options for VNC access to guest instances**

Configuration option=Default value	(Type) Description
novncproxy_base_url=http://127.0.0.1:6080/vnc_auto.html	(StrOpt) location of VNC console proxy, in the form "http://127.0.0.1:6080/vnc_auto.html"
vnc_enabled=true	(BoolOpt) enable VNC related features
vnc_keymap=en-us	(StrOpt) keymap for vnc
vnc_require_instance_uuid_as_password=false	(BoolOpt) When set to true, secure VNC connections by requiring the user to enter the instance uuid as the password. This ensures the user is connecting to the correct VNC console.
vncserver_listen=127.0.0.1	(StrOpt) IP address on which instance VNC servers 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 XCP VNC console proxy, in the form "http://127.0.0.1:6081/console"
xvpvncproxy_host=0.0.0.0	(StrOpt) Address that the XCP VNC proxy should bind to
xvpvncproxy_port=6081	(IntOpt) Port that the XCP VNC proxy should bind to



### Note

If you intend to support [live migration](#), you cannot specify a specific IP address for `vncserver_listen`, because that IP address will not exist on the destination host.



### Note

`vncserver_proxyclient_address` - Defaults to 127.0.0.1. This is the address of the compute host that nova will instruct proxies to use when connecting to instance servers. For all-in-one XenServer domU deployments this can be set to 169.254.0.1. For multi-host XenServer domU deployments this can be set to a dom0 management ip on the same network as the proxies. For multi-host libvirt deployments this can be set to a host management IP on the same network as the proxies.

## Accessing VNC Consoles with a Java client

To enable support for the OpenStack Java VNC client in Compute, we provide the `nova-xvpvncproxy` service, which you should run to enable this feature.

- `xvpvncproxy_port=[port]` - port to bind (defaults to 6081)

- `xvpvncproxy_host=[host]` - host to bind (defaults to 0.0.0.0)

As a client, you will need a special Java client, which is a version of TightVNC slightly modified to support our token auth:

```
$ git clone https://github.com/cloudbuilders/nova-xvpvncviewer
$ cd nova-xvpvncviewer
$ make
```

Then, to create a session, first request an access URL using `python-novaclient` and then run the client like so. To retrieve access URL:

```
$ nova get-vnc-console [server_id] xvpvnc
```

To run client:

```
$ java -jar VncViewer.jar [access_url]
```

## nova-novncproxy (novnc)

You will need the novnc package installed, which contains the nova-novncproxy service. As root:

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

The service should start automatically on install. To restart it:

```
# service novnc restart
```

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

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

In order to connect the service to your nova deployment, add the two following configuration options into your `nova.conf` file :

- `vncserver_listen=0.0.0.0`

This configuration option allow you to specify the address for the vnc service to bind on, make sure it is assigned one of the compute node interfaces. This address will be the one used by your domain file.

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



### Note

In order to have the live migration working, make sure to use the 0.0.0.0 address.

- `vncserver_proxyclient_address =127.0.0.1`

This is the address of the compute host that nova will instruct proxies to use when connecting to instance vncservers.

## Accessing a VNC console through a web browser

Retrieving an access\_url for a web browser is similar to the flow for the Java client. To retrieve the access URL:

```
$ nova get-vnc-console [server_id] novnc
```

Then, paste the URL into your web browser.

Additionally, you can use the OpenStack Dashboard (codenamed Horizon), to access browser-based VNC consoles for instances.

## Frequently asked questions about VNC access to VMs

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

A: nova-xvpvncproxy which ships with nova, is a new 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 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 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 will wait 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 would need 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 that novncproxy\_base\_url and xpvncproxy\_base\_url use a public ip; this is the url that is ultimately returned to clients, who generally will not have access to your private network. Your PROXYSERVER must be able to reach vncserver\_proxyclient\_address, as that is the address over which the vnc connection will be proxied.

See "Important nova-compute Options" for more information.

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

A: Make sure you have python-numpy installed, 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 horizon?**

A: These values are hard-coded in a Django HTML template. To alter them, you must edit the template file \_detail\_vnc.html. The location of this file will vary based on Linux distribution. On Ubuntu 12.04, the file can be found at /usr/share/pyshared/horizon/dashboards/nova/instances/templates/instances/\_detail\_vnc.html.

Modify the width and height parameters:

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

## Spice Console

OpenStack Compute has long had support for VNC consoles to guests. The VNC protocol is fairly limited, lacking support for multiple monitors, bi-directional audio, reliable cut+paste, video streaming and more. SPICE is a new protocol which aims to address all the limitations in VNC, to 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 using SPICE-over-websockets. The nova-spicehtml5proxy service communicates directly with the hypervisor process using SPICE.

Options for configuring SPICE as the console for OpenStack Compute can be found below.

**Table 15.2. Description of nova.conf [spice] section configuration options for SPICE HTML5 access to guest instances**

Configuration option=Default value	(Type) Description
html5proxy_base_url=http://\$nova-html5proxy_host:6082/spice_auto.html	(StrOpt) location of spice html5 console proxy, in the form "http://\$nova-html5proxy_host:6082/spice_auto.html"
enabled=false	(BoolOpt) enable spice related features
agent_enabled=true	(BoolOpt) enable spice guest agent support
keymap=en-us	(StrOpt) keymap for spice
server_listen=0.0.0.0	(StrOpt) IP address on which instance spice servers should listen

Configuration option=Default value	(Type) Description
server_proxyclient_address=\$compute_host	(StrOpt) Management IP Address on which instance spliceservers will listen on the compute host.

# 16. Security Hardening

## Table of Contents

Trusted Compute Pools ..... 301

OpenStack Compute can be integrated with various third-party technologies to increase security.

## Trusted Compute Pools

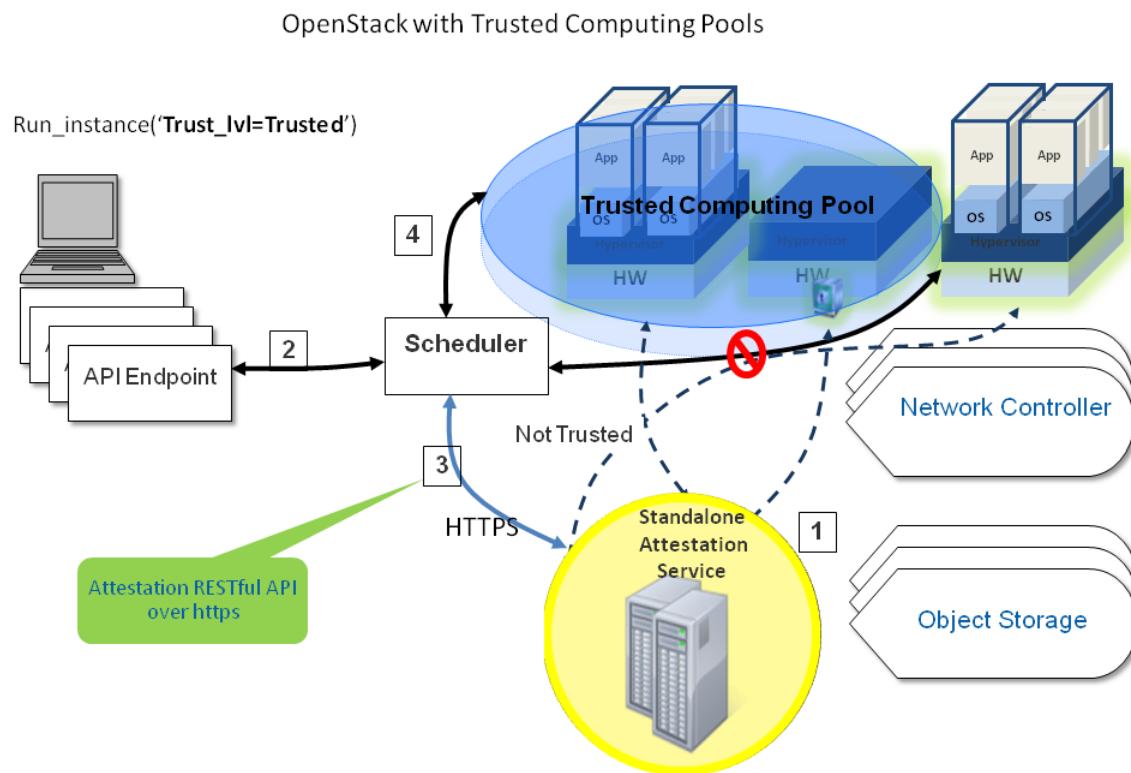
### Overview

Trusted compute pools enable administrators to designate a group of compute hosts as "trusted". These hosts use hardware-based security features, such as Intel's Trusted Execution Technology (TXT), to provide an additional level of security. Combined with an external standalone web-based remote attestation server, cloud providers can ensure that the compute node is running software with verified measurements, thus they can establish the foundation for the secure cloud stack. Through the Trusted Computing Pools, cloud subscribers can request services to be run on verified compute nodes.

The remote attestation server performs node verification through the following steps:

1. Compute nodes boot with Intel TXT technology enabled.
2. The compute node's BIOS, hypervisor and OS are measured.
3. These measured data is sent to the attestation server when challenged by attestation server.
4. The attestation server verifies those measurements against good/known database to determine nodes' trustworthiness.

A description of how to set up an attestation service is beyond the scope of this document. See the [Open Attestation](#) project for an open source project that can be used to implement an attestation service.



## Configuring the Compute service to use Trusted Compute Pools

The Compute service must be configured to use the connection information for the attestation service. The connection information is specified in the `trusted_computing` section of `nova.conf`. Specify the following parameters in this section.

<code>server</code>	Hostname or IP address of the host that runs the attestation service
<code>port</code>	HTTPS port for the attestation service
<code>server_ca_file</code>	Certificate file used to verify the attestation server's identity.
<code>api_url</code>	The attestation service URL path.
<code>auth_blob</code>	An authentication blob, which is required by the attestation service.

Add the following lines to `/etc/nova/nova.conf` in the `DEFAULT` and `trusted_computing` sections to enable scheduling support for Trusted Compute Pools, and edit the details of the `trusted_computing` section based on the details of your attestation service.

```
[DEFAULT]
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler
scheduler_available_filters=nova.scheduler.filters.all_filters
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter,
TrustedFilter

[trusted_computing]
server=10.1.71.206
port=8443
server_ca_file=/etc/nova/ssl.10.1.71.206.crt
# If using OAT v1.5, use this api_url:
api_url=/AttestationService/resources
# If using OAT pre-v1.5, use this api_url:
#api_url=/OpenAttestationWebServices/V1.0
auth_blob=i-am-openstack
```

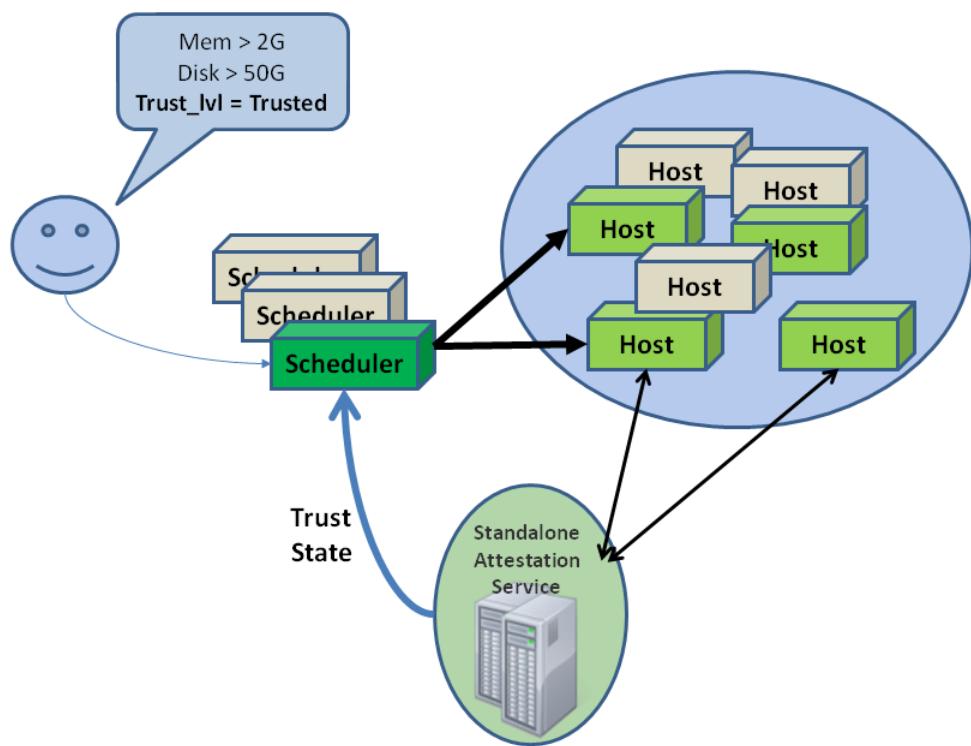
Restart the nova-compute and nova-scheduler services after making these changes.

## Specify trusted flavors

One or more flavors must be configured as "trusted". Users can then request trusted nodes by specifying one of these trusted flavors when booting a new instance. Use the **nova-manage instance\_type set\_key** command to set a flavor as trusted. For example, to set the m1.tiny flavor as trusted:

```
# nova-manage instance_type set_key m1.tiny trust:trusted_host trusted
```

A user can request that their instance runs on a trusted host by specifying a trusted flavor when invoking the **nova boot** command.



# 17. OpenStack Compute Automated Installations

## Table of Contents

Deployment Tool for OpenStack using Puppet (dodai-deploy) ..... 305

In a large-scale cloud deployment, automated installations are a requirement for successful, efficient, repeatable installations. Automation for installation also helps with continuous integration and testing. This chapter offers some tested methods for deploying OpenStack Compute with either Puppet (an infrastructure management platform) or Chef (an infrastructure management framework) paired with Vagrant (a tool for building and distributing virtualized development environments).

## Deployment Tool for OpenStack using Puppet (dodai-deploy)

The dodai-deploy is a software management tool. It supports the following softwares.

- OpenStack Folsom(Compute, Glance, Swift, Keystone). Compute includes Nova, Horizon, Cinder and Quantum.
- hadoop 0.22.0
- sun grid engine 6.2u5

## Features

- Manage installation, uninstallation and testing of a software.
- Support deployment on multiple machines.
- Support target machines in different network segments.
- Provide web UI to facilitate user operations.
- Provide REST API to make it possible to integrate it with other tools.
- Support parallel installation of software components.

## OSes supported

**Table 17.1. OSes supported**

	ubuntu 12.04		)
OpenStack Folsom (Compute, Glance, Swift, Keystone)			)

	ubuntu 12.04		
hadoop 0.22.0	:)	:)	:)
sun grid engine 6.2u5	:)	:)	:)

## Glossary

- **dodai-deploy server** - The server in which services of dodai-deploy is started.
- **Node** - The machine that is the target of installation.
- **Nova, Glance, Swift etc.**
- **Proposal** - The set of the kinds of configurations which describe how to install a software. The configurations include "Node config", "Config item", "Software config", "Component config".
- **Node config** - A configuration that describes which component to be installed on a node.
- **Config item** - A variable which can be used in the content of software config and component config.
- **Software config** - A configuration that describes the content of a configuration file for all components.
- **Component config** - A configuration that describes the content of a configuration file for only one component.

## Installation

The \$home in the following sections is the path of the home directory of the dodai-deploy.

1. Download dodai-deploy.

Execute the following commands on the dodai-deploy server and all the nodes.

```
$ sudo apt-get install git -y
$ git clone https://github.com/nii-cloud/dodai-deploy.git
$ cd dodai-deploy
```

2. Set up the dodai-deploy server.

Execute the following commands on dodai-deploy server to install necessary softwares and modify their settings.

```
$ sudo $home/setup-env/setup.sh server
```

3. Set up nodes.

Execute the following commands on all the nodes to install necessary softwares and modify their settings.

```
$ sudo $home/setup-env/setup.sh -s $server node
```

The \$server in the above command is the fully qualified domain name (fqdn) of the dodai-deploy server. You can confirm the fqdn with the following command.

```
$ sudo hostname -f
```

After nodes were set up, the system time of nodes should be synchronized with dodai-deploy server.

#### 4. Set up storage device for Swift.

You must set up a storage device before swift is installed. You should execute the commands for a physical device or for a loopback device on all nodes in which swift storage server is to be installed.

- For a physical device, use the following command.

```
$ sudo $home/setup-env/setup-storage-for-swift.sh physical $storage_path  
$storage_dev
```

For example,

```
$ sudo $home/setup-env/setup-storage-for-swift.sh physical /srv/node sdb1
```

- For a loopback device, use the following command.

```
$ sudo $home/setup-env/setup-storage-for-swift.sh loopback $storage_path  
$storage_dev $size
```

For example,

```
$ sudo $home/setup-env/setup-storage-for-swift.sh loopback /srv/node sdb1  
4
```

#### 5. Create volume group for nova-volume.

You must create a volume group before nova-volume is installed. You should execute the commands for a physical device or for a loopback device on the node in which nova-volume is to be installed.

- For a physical device, use the following command.

```
$ sudo $home/setup-env/create-volume-group.sh physical $volume_group_name  
$device_path
```

For example,

```
$ sudo $home/setup-env/create-volume-group.sh physical nova-volumes /dev/sdb1
```

- For a loopback device, use the following command.

```
$ sudo $home/setup-env/create-volume-group.sh loopback $volume_group_name  
$file_path $size
```

For example,

```
$ sudo $home/setup-env/create-volume-group.sh loopback nova-volumes /root/  
volume.data 4
```

## 6. Start servers.

Execute the following command on the *dodai-deploy* server to start the web server and job server.

```
$ sudo $home/script/start-servers production
```

You can stop the web server and job server with the following command.

```
$ sudo $home/script/stop-servers
```

## Using web UI

You can find step-by-step guidance at [http://\\$dodai\\_deploy\\_server:3000/](http://$dodai_deploy_server:3000/).

## Using REST APIs

An API simulator can be found at [http://\\$dodai\\_deploy\\_server:3000/rest\\_apis/index.html](http://$dodai_deploy_server:3000/rest_apis/index.html). You can get the list of REST APIs with it. You can also execute APIs by simply filling in parameters and clicking the "Execute" button.

## Notes

### 1. SSH login nova instance after test of nova

An instance will be started during the test of nova. After the test, you can login the instance by executing the following commands.

For openstack nova folsom,

```
$ sudo -i  
$ cd /var/lib/nova  
$ . novarc  
$ euca-describe-instances  
$ ssh -i mykey.priv 10.0.0.3
```

2. Glance should be installed before using nova, because nova depends on glance in default settings.

In `/etc/nova/nova.conf` the value of setting `image_service` is `nova.image.GlanceImageService`.

3. Change Linux's setting `net.ipv4.ip_forward` to 1 in the machine where nova-network will be installed before nova installation with the following command.

```
$ sudo sysctl -w net.ipv4.ip_forward=1
```

You can recover the setting with the following command.

```
$ sudo sysctl -w net.ipv4.ip_forward=0
```

# 18. OpenStack Compute Tutorials

## Table of Contents

Running Your First Elastic Web Application on the Cloud ..... 310

We want OpenStack to make sense, and sometimes the best way to make sense of the cloud is to try out some basic ideas with cloud computing. Flexible, elastic, and scalable are a few attributes of cloud computing, so these tutorials show various ways to use virtual computing or web-based storage with OpenStack components.

## Running Your First Elastic Web Application on the Cloud

In this OpenStack Compute tutorial, we'll walk through the creation of an elastic, scalable cloud running a WordPress installation on a few virtual machines.

The tutorial assumes you have obtained a TryStack account at <http://trystack.org>. It has a working installation of OpenStack Compute, or you can install your own using the installation guides.

We'll go through this tutorial in parts:

- Setting up a user on the TryStack cloud.
- Getting images for your application servers.
- On the instances you spin up, installing Wordpress and its dependencies, the Memcached plugin, and multiple memcache servers.

### Part I: Setting Up as a TryStack User

In this part, we'll get a TryStack account using our Facebook login. Onward, brave cloud pioneers!

Go to the TryStack Facebook account at <https://www.facebook.com/groups/269238013145112/> and request to join the group.

Once you've joined the group, go to the TryStack dashboard and click **Login using Facebook**.

Enter your Facebook login information to receive your username and password that you can use with the Compute API.

Next, install the python-novaclient and set up your environment variables so you can use the client with your username and password already entered. Here's what works well on Mac OS X.

```
$ pip install -e git+https://github.com/openstack/python-novaclient.git#egg=python-novaclient
```

Next, create a file named `openrc` to contain your TryStack credentials, such as:

```
export OS_USERNAME=joecool
export OS_PASSWORD=coolword
export OS_TENANT_NAME=coolu
export OS_AUTH_URL=http://trystack.org:5000/v2.0
export NOVA_VERSION=1.1
```

Lastly, run this file to source your credentials.

```
$ source openrc
```

You can always retrieve your username and password from [https://trystack.org/dash/api\\_info/](https://trystack.org/dash/api_info/) after logging in with Facebook.

Okay, you've created the basic scaffolding for your cloud user so that you can get some images and run instances on TryStack with your starter set of StackDollars. You're rich, man! Now to Part II!

## Part II: Starting Virtual Machines

Understanding what you can do with cloud computing means you should have a grasp on the concept of virtualization. With virtualization, you can run operating systems and applications on virtual machines instead of physical computers. To use a virtual machine, you must have an image that contains all the information about which operating system to run, the user login and password, files stored on the system, and so on. Fortunately, TryStack provides images for your use.

Basically, run:

```
$ nova image-list
```

and look for the images available in the text that returns. Look for the ID value.

ID	Name	Status	Server
12	natty-server-cloudimg-amd64-kernel	ACTIVE	
13	natty-server-cloudimg-amd64	ACTIVE	
14	oneiric-server-cloudimg-amd64-kernel	ACTIVE	
15	oneiric-server-cloudimg-amd64	ACTIVE	

Now get a list of the flavors you can launch:

```
$ nova flavor-list
```

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

Create a keypair to launch the image, in a directory where you run the nova boot command later.

```
$ nova keypair-add mykeypair > mykeypair.pem
```

Create security group that enables public IP access for the webserver that will run WordPress for you. You can also enable port 22 for SSH.

```
$ nova secgroup-create openpub "Open for public"
$ nova secgroup-add-rule openpub icmp -1 -1 0.0.0.0/0
$ nova secgroup-add-rule openpub tcp 22 22 0.0.0.0/0
```

Next, with the ID value of the server you want to launch and the ID of the flavor you want to launch, use your credentials to start up the instance with the identifier you got by looking at the image list.

```
$ nova boot --image 15 --flavor 2 --key_name mykeypair --security_groups
openpub testtutorial
```

Property	Value
accessIPv4	
accessIPv6	
adminPass	StuacCpAr7evnz5Q
config_drive	
created	2012-03-21T20:31:40Z
flavor	m1.small
hostId	
id	1426
image	oneiric-server-cloudimg-amd64
key_name	testkey2
metadata	{}
name	testtut
progress	0
status	BUILD
tenant_id	296
updated	2012-03-21T20:31:40Z
user_id	facebook521113267

	uuid		be9f80e8-7b20-49e8-83cf-fa059a36c9f8	
--	------	--	--------------------------------------	--

Now you can look at the state of the running instances by using `nova list`.

```
$ nova list
```

+-----+	+-----+	+-----+	+-----+
ID	Name	Status	Networks
1426   testtut   ACTIVE   internet=8.22.27.251			

The instance goes from “launching” to “running” in a short time, and you should be able to connect via SSH. Look at the IP addresses so that you can connect to the instance once it starts running.

## Diagnose your compute node

You can obtain extra informations about the instance you just spawned : its CPU usage, the memory, the disk io or network io, per instance, by running the **nova diagnostics** command:

```
$ nova list
```

+-----+	+-----+	+-----+
ID	Name	Status   Networks
35	test   ACTIVE   local-net=192.168.4.	

```
$ nova diagnostics 50191b9c-b26d-4b61-8404-f149c29acd5a
```

+-----+	+-----+
Property	Value
cpu0_time	9160000000
memory	524288
memory-actual	524288
memory-rss	178040
vda_errors	-1
vda_read	3146752
vda_read_req	202
vda_write	1024
vda_write_req	1
vnet0_rx	610
vnet0_rx_drop	0
vnet0_rx_errors	0

vnet0_rx_packets	7
vnet0_tx	0
vnet0_tx_drop	0
vnet0_tx_errors	0
vnet0_tx_packets	0

## Part III: Installing the Needed Software for the Web-Scale Scenario

Basically launch a terminal window from any computer, and enter:

```
$ ssh -i mykeypair ubuntu@10.127.35.119
```

On this particular image, the 'ubuntu' user has been set up as part of the sudoers group, so you can escalate to 'root' via the following command:

```
$ sudo -i
```

### On the first VM, install WordPress

Now, you can install WordPress. Create and then switch to a blog directory:

```
$ mkdir blog  
$ cd blog
```

Download WordPress directly to you by using wget:

```
$ wget http://wordpress.org/latest.tar.gz
```

Then unzip the package using:

```
$ tar -xzvf latest.tar.gz
```

The WordPress package will extract into a folder called wordpress in the same directory that you downloaded latest.tar.gz.

Next, enter "exit" and disconnect from this SSH session.

### On a second VM, install MySQL

Next, SSH into another virtual machine and install MySQL and use these instructions to install the WordPress database using the MySQL Client from a command line: [Using the MySQL Client - Wordpress Codex](#).

## On a third VM, install Memcache

Memcache makes Wordpress database reads and writers more efficient, so your virtual servers can go to work for you in a scalable manner. SSH to a third virtual machine and install Memcache:

```
$ apt-get install memcached
```

## Configure the Wordpress Memcache plugin

From a web browser, point to the IP address of your Wordpress server. Download and install the Memcache Plugin. Enter the IP address of your Memcache server.

## Running a Blog in the Cloud

That's it! You're now running your blog on a cloud server in OpenStack Compute, and you've scaled it horizontally using additional virtual images to run the database and Memcache. Now if your blog gets a big boost of comments, you'll be ready for the extra reads-and-writes to the database.

# 19. Support

## Table of Contents

Community Support ..... 316

Online resources aid in supporting OpenStack and the community members are willing and able to answer questions and help with bug suspicions. We are constantly improving and adding to the main features of OpenStack, but if you have any problems, do not hesitate to ask. Here are some ideas for supporting OpenStack and troubleshooting your existing installations.

## Community Support

Here are some places you can locate others who want to help.

### [ask.openstack.org](http://ask.openstack.org)

During setup or testing, you may have questions about how to do something, or end up in a situation where you can't seem to get a feature to work correctly. The [ask.openstack.org](http://ask.openstack.org) site is available for questions and answers. When visiting the Ask site at <http://ask.openstack.org>, it is usually good to at least scan over recently asked questions to see if your question has already been answered. If that is not the case, then proceed to adding a new question. Be sure you 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, link to screenshots, and so on.

### OpenStack mailing lists

Posting your question or scenario to the OpenStack mailing list is a great way to get answers and insights. You can learn from and help others who may have the same scenario as you. Go to <https://launchpad.net/~openstack> and click "Subscribe to mailing list" or view the archives at <https://lists.launchpad.net/openstack/>. You may be interested in the other mailing lists for specific projects or development - these can be found [on the wiki](#). A description of all the additional mailing lists is available at <http://wiki.openstack.org/MailingLists>.

### The OpenStack Wiki search

The [OpenStack wiki](#) contains content on a broad range of topics, but some of it sits a bit below the surface. Fortunately, the wiki search feature is very powerful in that it can do both searches by title and by content. If you are searching for specific information, say about "networking" or "api" for nova, you can find lots of content using the search feature. More is being added all the time, so be sure to check back often. You can find the search box in the upper right hand corner of any OpenStack wiki page.

## The Launchpad Bugs area

So you think you've found a bug. That's great! Seriously, it is. The OpenStack community values your setup and testing efforts and wants your feedback. To log a bug you must have a Launchpad account, so sign up at <https://launchpad.net/+login> if you do not already have a Launchpad ID. You can view existing bugs and report your bug in the Launchpad Bugs area. It is suggested that you first use the search facility to see if the bug you found has already been reported (or even better, already fixed). If it still seems like your bug is new or unreported then it is time to 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, link to screenshots, etc.
- Be sure to include what version of the software you are using. This is especially critical if you are using a development branch eg. "Grizzly release" vs git commit `bc79c3ecc55929bac585d04a03475b72e06a3208`.
- Any deployment specific info is helpful as well, such as Ubuntu 12.04, multi-node install.

The Launchpad Bugs areas are available here - :

- OpenStack Compute: <https://bugs.launchpad.net/nova>
- OpenStack Object Storage: <https://bugs.launchpad.net/swift>
- OpenStack Image Delivery and Registration: <https://bugs.launchpad.net/glance>
- OpenStack Identity: <https://bugs.launchpad.net/keystone>
- OpenStack Dashboard: <https://bugs.launchpad.net/horizon>
- OpenStack Network Connectivity: <https://bugs.launchpad.net/quantum>

## The OpenStack IRC channel

The OpenStack community lives and breathes in the #openstack IRC channel on the Freenode network. You can come by to hang out, ask questions, or get immediate feedback for urgent and pressing issues. To get into the IRC channel you need to install an IRC client or use a browser-based client by going to <http://webchat.freenode.net/>. You can also use Colloquy (Mac OS X, <http://colloquy.info/>) or 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 you can then paste into the channel. The OpenStack IRC channel is: #openstack on irc.freenode.net. A list of all the OpenStack-related IRC channels is at <https://wiki.openstack.org/wiki/IRC>.

# 20. Troubleshooting OpenStack Compute

## Table of Contents

Log files for OpenStack Compute .....	318
Common Errors and Fixes for OpenStack Compute .....	318
Manually reset the state of an instance .....	319
Problems with Injection .....	320

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 running on that node. Another common problem is trying to run 32-bit images on a 64-bit compute node. This section offers more information about how to troubleshoot Compute.

## Log files for OpenStack Compute

Log files are stored in /var/log/nova and there is a log file for each service, for example nova-compute.log. You can format the log strings using options for the nova.log module. The options used to set format strings are: logging\_context\_format\_string and 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, and then the nova daemon logs to syslog.

## Common Errors and Fixes for OpenStack 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. We are constantly fixing bugs, so online resources are a great way to get the most up-to-date errors and fixes.

Credential errors, 401, 403 forbidden errors

A 403 forbidden error is caused by missing credentials. Through current installation methods, there are basically two ways to get the novarc file. The manual method requires getting it from within a project zipfile, and the scripted method just generates novarc out of the project zip file and sources it for you. If you do the manual method through a zip file, then the following novarc alone, you end up losing the creds that are tied to the user you created with nova-manage in the steps before.

When you run nova-api the first time, it generates the certificate authority information, including openssl.cnf. If it gets started out of order, you may not be able to create your zip

file. Once your CA information is available, you should be able to go back to nova-manage to create your zipfile.

You may also need to check your proxy settings to see if they are causing problems with the 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 using flat manager networking, you do not have a dhcp server, and an ami-tiny image doesn't support interface injection so you cannot connect to it. The fix for this type of problem is to use an Ubuntu image, which should obtain an IP address correctly with FlatManager network settings. To troubleshoot other possible problems with an instance, such as one that stays in a spawning state, first check your instances directory for i-ze0bnh1q dir to make sure it has the following files:

- libvirt.xml
- disk
- disk-raw
- kernel
- ramdisk
- console.log (Once the instance actually starts you should see a console.log.)

Check the file sizes to see if they are reasonable. If any are missing/zero/very small then nova-compute has somehow not completed download of the images from objectstore.

Also check nova-compute.log for exceptions. Sometimes they don't show up in the console output.

Next, check the /var/log/libvirt/qemu/i-ze0bnh1q.log file to see if it exists and has any useful error messages in it.

Finally, from the instances/i-ze0bnh1q directory, try `virsh create libvirt.xml` and see if you get an error there.

## Manually reset the state of an instance

If an instance gets stuck in an intermediate state (e.g., "deleting"), you can manually reset the state of an instance using the **nova reset-state** command. This will reset it to an error state, which you can then delete. For example:

```
$ nova reset-state c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
$ nova delete c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

You can also use the `--active` to force the instance back into an active state instead of an error state, for example:

```
$ nova reset-state --active c6bbbf26-b40a-47e7-8d5c-eb17bf65c485
```

## Problems with Injection

If you are diagnosing problems with instances not booting, or booting slowly, consider investigating file injection as a cause. Setting `libvirt_injection_partition` to `-2` disables injection in libvirt. This can be required if you want to make user specified files available from the metadata server (and config drive is not enabled), for performance reasons, and also to avoid boot failure if injection itself fails.

## 21. Acknowledgements

This document is the work of a large collection of people, from many different organisations who have put in long hours to create what you read here.

A few in particular deserve special mention:

- Rackspace - who created much of the original documentation, on which the current versions are based.
- CSS Corp - who created the original version of the image customisation guide.
- David Cramer - who develops the documentation toolchain.
- Anne Gentle - without her stewardship, these documents would not exist.