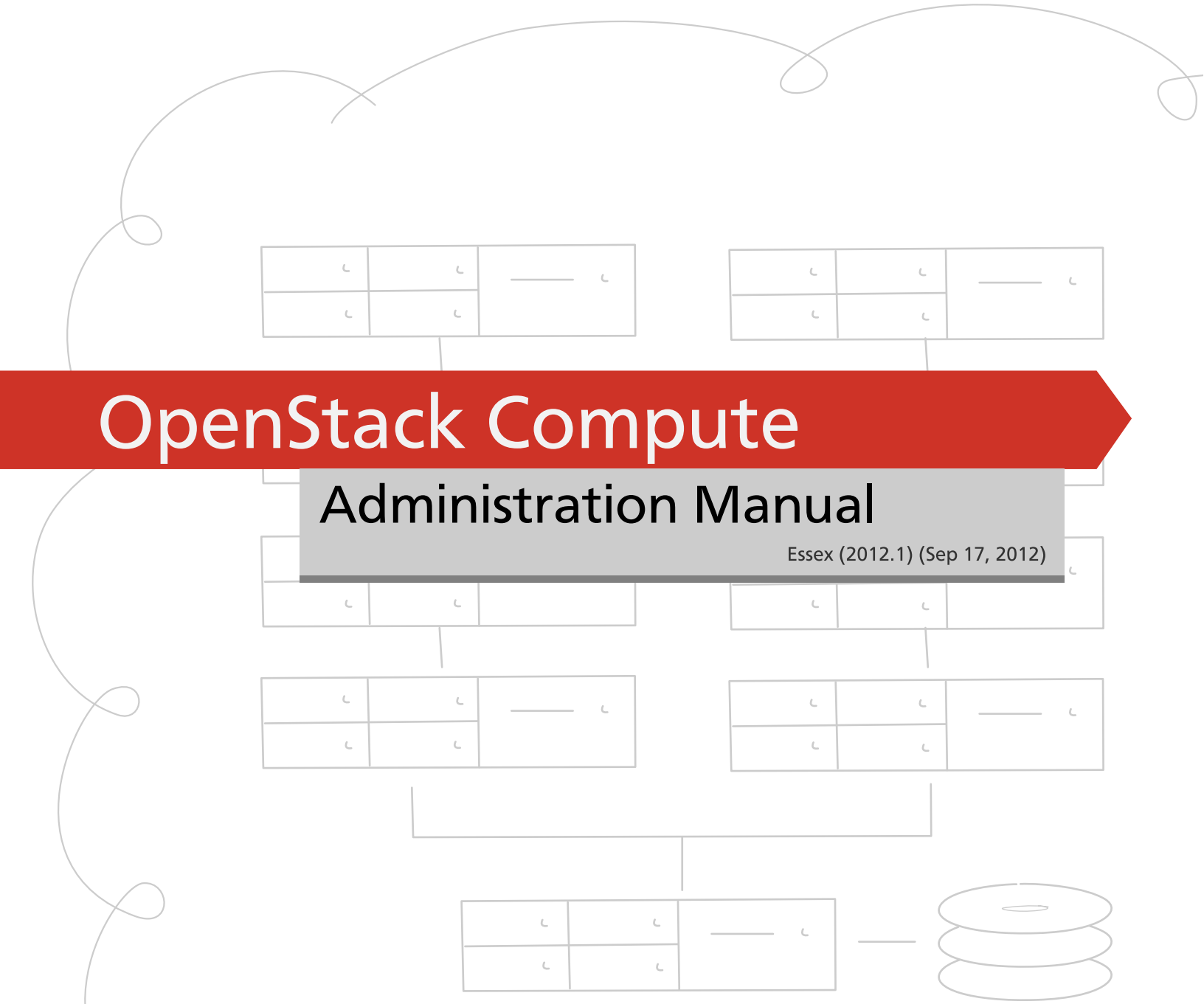
OpenStack Compute

Administration Manual

Essex (2012.1) (Sep 17, 2012)



openstack™

docs.openstack.org

OpenStack Compute Administration Manual

Essex (2012.1) (2012-09-17)

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.

Table of Contents

1. Getting Started with OpenStack 1
What is OpenStack? 1
Components of OpenStack 1
OpenStack Project Architecture Overview 2
Conceptual Architecture 3
Logical Architecture 4
Dashboard 5
Compute 6
Object Store 7
Image Store 8
Identity 8
Future Projects 9
Why Cloud? 9
2. Introduction to OpenStack Compute 11
Hypervisors 11
Users and Projects (Tenants) 11
Images and Instances 12
System Architecture 15
Block Storage and OpenStack Compute 15
3. Installing OpenStack Compute 17
Compute and Image System Requirements 17
Example Installation Architectures 18
Service Architecture 19
Installing OpenStack Compute on Debian 20
Installing on Fedora or Red Hat Enterprise Linux 6 21
Installing on Ubuntu 22
ISO Distribution Installation 22
Scripted Installation 22
Manual Installation on Ubuntu 22
Installing on Citrix XenServer 23
4. Configuring OpenStack Compute 24
Post-Installation Configuration for OpenStack Compute 24
Setting Configuration Options in the `nova.conf` File 24
Setting Up OpenStack Compute Environment on the Compute Node 25
Creating Credentials 26
Creating Certificates 27
Enabling Access to VMs on the Compute Node 27
Configuring Multiple Compute Nodes 28
Determining the Version of Compute 30
Diagnose your compute nodes 30
General Compute Configuration Overview 30
Example `nova.conf` Configuration Files 31
Configuring Logging 37
Configuring Hypervisors 38
Configuring Authentication and Authorization 39
Configuring Compute to use IPv6 Addresses 42
Configuring Image Service and Storage for Compute 43
Configuring Migrations 43

Enabling true live migration 47
Installing MooseFS as shared storage for the instances directory 47
Installing the MooseFS metadata and metalogger servers 48
Installing the MooseFS chunk and client services 50
Access to your cluster storage 51
Configuring Database Connections 52
Configuring the Compute Messaging System 53
Configuration for RabbitMQ 53
Configuration for Qpid 54
Common Configuration for Messaging 55
Configuring the Compute API 55
Configuring the EC2 API 57
5. Reference for Configuration Options in nova.conf 59
6. Identity Management 75
Basic Concepts 75
User management 77
Service management 81
Configuration File 81
Sample Configuration Files 82
Running 82
Migrating from legacy versions of keystone 82
Step 1: Configure keystone.conf 82
Step 2: db_sync your new, empty database 82
Step 3: Import your legacy data 82
Step 4: Import your legacy service catalog 83
Migrating from Legacy Authentication 83
Step 1: Export your data from Compute 83
Step 2: db_sync your new, empty database 83
Step 3: Import your data to Keystone 83
Initializing Keystone 83
Adding Users, Tenants, and Roles with python-keystoneclient 84
Token Auth Method 84
Password Auth Method 84
Example usage 84
Tenants 85
Users 86
Roles 88
Services 89
Configuring Services to work with Keystone 90
Setting up credentials 90
Setting up services 91
Setting Up Middleware 92
7. Image Management 98
Getting virtual machine images 98
CirrOS (test) images 98
Ubuntu images 98
Fedora images 99
OpenSUSE and SLES 11 images 99
Rackspace Cloud Builders (multiple distros) images 99
Tool support for creating images 99
Oz (KVM) 99

VMBuilder (KVM, Xen) ... 99
VeeWee (KVM) ... 99
Creating raw or QCOW2 images ... 99
Booting a test image ... 104
Tearing down (deleting) Instances ... 106
Pausing and Suspending Instances ... 106
Pausing instance ... 107
Suspending instance ... 107
Select a specific node to boot instances on ... 107
Image management ... 107
Creating a Linux Image – Ubuntu & Fedora ... 108
Creating a Windows Image ... 113
Creating images from running instances with KVM and Xen ... 114
8. Hypervisors ... 117
Selecting a Hypervisor ... 117
Hypervisor Configuration Basics ... 117
KVM ... 118
Checking for hardware virtualization support ... 119
Enabling KVM ... 120
Troubleshooting ... 120
QEMU ... 121
Tips and fixes for QEMU on RHEL ... 121
Xen, XenAPI, XenServer and XCP ... 122
Xen terminology ... 122
XenAPI deployment architecture ... 123
XenAPI pools ... 124
Installing XenServer and XCP ... 125
Further reading ... 126
LXC (Linux containers) ... 126
VMware ESX/ESXi Server Support ... 126
Introduction ... 126
Prerequisites ... 127
Configure Tomcat to serve WSDL files ... 128
VMWare configuration options ... 128
9. Networking ... 129
Networking Options ... 129
DHCP server: dnsmasq ... 131
Metadata service ... 132
Configuring Networking on the Compute Node ... 133
Configuring Flat Networking ... 134
Configuring Flat DHCP Networking ... 137
Outbound Traffic Flow with Any Flat Networking ... 140
Configuring VLAN Networking ... 141
Cloudpipe — Per Project Vpns ... 148
Enabling Ping and SSH on VMs ... 159
Configuring Public (Floating) IP Addresses ... 160
Private and Public IP Addresses ... 160
Enabling IP forwarding ... 160
Creating a List of Available Floating IP Addresses ... 161
Adding a Floating IP to an Instance ... 162
Automatically adding floating IPs ... 162

Removing a Network from a Project ... 162
Using multiple interfaces for your instances (multinic) ... 163
Using the multinic feature ... 165
Existing High Availability Options for Networking ... 166
Troubleshooting Networking ... 169
10. Volumes ... 172
Managing Volumes ... 172
Volume drivers ... 184
Ceph RADOS block device (RBD) ... 184
Nexenta ... 184
Using the XenAPI Storage Manager Volume Driver ... 185
Boot From Volume ... 187
11. Scheduling ... 190
Filter Scheduler ... 190
Filters ... 190
AllHostsFilter ... 191
AvailabilityZoneFilter ... 192
ComputeFilter ... 192
CoreFilter ... 192
DifferentHostFilter ... 192
IsolatedHostsFilter ... 193
JsonFilter ... 193
RamFilter ... 194
SameHostFilter ... 194
SimpleCIDRAffinityFilter ... 195
Costs and Weights ... 196
nova.scheduler.least_cost.compute_fill_first_cost_fn ... 197
nova.scheduler.least_cost.noop_cost_fn ... 197
Other Schedulers ... 197
Chance Scheduler ... 197
Multi Scheduler ... 197
Simple Scheulder ... 197
12. System Administration ... 199
Understanding the Compute Service Architecture ... 200
Managing Compute Users ... 201
Managing the Cloud ... 201
Using Migration ... 203
Nova Disaster Recovery Process ... 205
13. OpenStack Interfaces ... 209
About the Dashboard ... 209
System Requirements for the Dashboard ... 209
Installing the OpenStack Dashboard ... 209
Configuring the Dashboard ... 210
Validating the Dashboard Install ... 211
Launching Instances using Dashboard ... 212
Overview of VNC Proxy ... 215
About nova-consoleauth ... 215
Typical Deployment ... 215
Frequently asked questions about VNC access to VMs ... 218
14. OpenStack Compute Automated Installations ... 221
Deployment Tool for OpenStack using Puppet (dodai-deploy) ... 221

15. OpenStack Compute Tutorials 226
Running Your First Elastic Web Application on the Cloud 226
Part I: Setting Up as a TryStack User 226
Part II: Starting Virtual Machines 227
Diagnose your compute node 229
Part III: Installing the Needed Software for the Web-Scale Scenario 229
Running a Blog in the Cloud 230
16. Support and Troubleshooting 231
Community Support 231
Troubleshooting OpenStack Object Storage 232
Handling Drive Failure 232
Handling Server Failure 233
Detecting Failed Drives 233
Troubleshooting OpenStack Compute 233
Log files for OpenStack Compute 233
Common Errors and Fixes for OpenStack Compute 234

List of Figures

2.1. Base image state with no running instances 13
2.2. Instance creation from image and run time state 14
2.3. End state of image and volume after instance exits 14
4.1. KVM, FlatDHCP, MySQL, Glance, LDAP, and optionally sheepdog 34
4.2. KVM, Flat, MySQL, and Glance, OpenStack or EC2 API 36
4.3. KVM, Flat, MySQL, and Glance, OpenStack or EC2 API 37
4.4. MooseFS deployment for OpenStack 48
9.1. Flat network, all-in-one server installation 135
9.2. Flat network, single interface, multiple servers 136
9.3. Flat network, multiple interfaces, multiple servers 136
9.4. Flat DHCP network, multiple interfaces, multiple servers with libvirt driver 138
9.5. Flat DHCP network, multiple interfaces, multiple servers, network HA with XenAPI driver 139
9.6. Single adaptor hosts, first route 140
9.7. Single adaptor hosts, second route 141
9.8. VLAN network, multiple interfaces, multiple servers, network HA with XenAPI driver 146
9.9. Configuring Viscosity 157
9.10. multinic flat manager 163
9.11. multinic flatdhcp manager 164
9.12. multinic VLAN manager 165
9.13. High Availability Networking Option 167
11.1. Filtering 191
11.2. Computing weighted costs 196

List of Tables

3.1. Hardware Recommendations 17
4.1. Description of nova.conf log file configuration options 37
4.2. Description of nova.conf file configuration options for hypervisors 38
4.3. Description of nova.conf configuration options for authentication 39
4.4. Description of nova.conf file configuration options for credentials (crypto) 40
4.5. Description of nova.conf file configuration options for LDAP 41
4.6. Description of nova.conf configuration options for IPv6 42
4.7. Description of nova.conf file configuration options for S3 access to image storage 43
4.8. Description of nova.conf file configuration options for live migration 46
4.9. Description of nova.conf configuration options for databases 52
4.10. Description of `nova.conf` configuration options for Remote Procedure Calls and RabbitMQ Messaging 53
4.11. Description of `nova.conf` configuration options for Tuning RabbitMQ Messaging 53
4.12. Remaining `nova.conf` configuration options for Qpid support 54
4.13. Description of `nova.conf` configuration options for Customizing Exchange or Topic Names 55
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 59
5.2. Description of nova.conf configuration options for databases 62
5.3. Description of nova.conf configuration options for IPv6 63
5.4. Description of nova.conf log file configuration options 63
5.5. Description of nova.conf file configuration options for nova- services 63
5.6. Description of nova.conf file configuration options for credentials (crypto) 64
5.7. Description of nova.conf file configuration options for policies (policy.json) 64
5.8. Description of nova.conf file configuration options for quotas 64
5.9. Description of nova.conf file configuration options for testing purposes 65
5.10. Description of nova.conf configuration options for authentication 65
5.11. Description of nova.conf file configuration options for LDAP 66
5.12. Description of nova.conf file configuration options for roles and authentication 67
5.13. Description of nova.conf file configuration options for EC2 API 67
5.14. Description of nova.conf file configuration options for VNC access to guest instances 67
5.15. Description of nova.conf file configuration options for networking options 68
5.16. Description of nova.conf file configuration options for live migration 69
5.17. Description of nova.conf file configuration options for compute nodes 69
5.18. Description of nova.conf file configuration options for bare metal deployment 70
5.19. Description of nova.conf file configuration options for hypervisors 70
5.20. Description of nova.conf file configuration options for console access to VMs on VMWare VMRC or XenAPI 71
5.21. Description of nova.conf file configuration options for S3 access to image storage 72

5.22. Description of nova.conf file configuration options for schedulers that use algorithms to assign VM launch on particular compute hosts 72
5.23. Description of nova.conf file configuration options for volumes attached to VMs 73
6.1. Description of keystone.conf file configuration options for LDAP 96
8.1. Description of nova.conf configuration options for the compute node 117
11.1. Description of Simple Scheduler configuration options 198
14.1. OSes supported 221

1. Getting Started with OpenStack

OpenStack is a collection of open source technology that provides massively scalable open source cloud computing software. Currently OpenStack develops two related projects: OpenStack Compute, which offers computing power through virtual machine and network management, and OpenStack Object Storage which is software for redundant, scalable object storage capacity. Closely related to the OpenStack Compute project is the Image Service project, named Glance. 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.

What is OpenStack?

OpenStack offers open source software to build public and private clouds. OpenStack is a community and a project as well as open source software to help organizations run clouds for virtual computing or storage. OpenStack contains a collection of open source projects that are community-maintained including OpenStack Compute (code-named Nova), OpenStack Object Storage (code-named Swift), and OpenStack Image Service (code-named Glance). OpenStack provides an operating platform, or toolkit, for orchestrating clouds.

OpenStack is more easily defined once the concepts of cloud computing become apparent, but we are 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/~openstackso it can be updated or removed.

Components of OpenStack

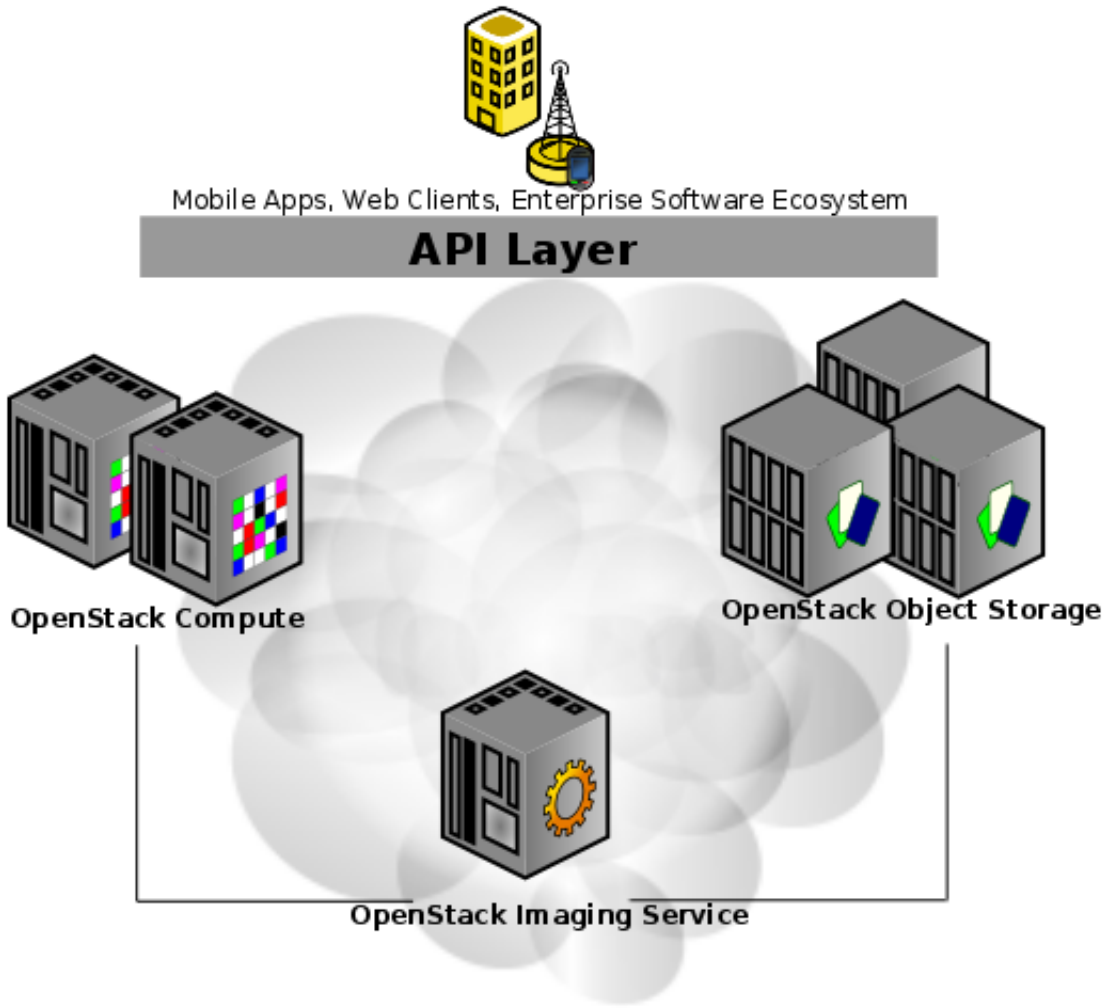
There are currently three main components of OpenStack: Compute, Object Storage, and Image Service. Let's look at each in turn.

OpenStack Compute is a cloud fabric controller, used to start up virtual instances for either a user or a group. It's also used to configure networking for each instance or project that contains multiple instances for a particular project.

OpenStack Object Storage is a system to store objects in a massively scalable large capacity system with built-in redundancy and failover. Object Storage has a variety of applications, such as backing up or archiving data, serving graphics or videos (streaming data to a userâ €™s browser), storing secondary or tertiary static data, developing new applications with data storage integration, storing data when predicting storage capacity is difficult, and creating the elasticity and flexibility of cloud-based storage for your web applications.

OpenStack Image Service is a lookup and retrieval system for virtual machine images. It can be configured in three ways: using OpenStack Object Store to store images; using Amazon's Simple Storage Solution (S3) storage directly; or using S3 storage with Object Store as the intermediate for S3 access.

The following diagram shows the basic relationships between the projects, how they relate to each other, and how they can fulfill the goals of open source cloud computing.



OpenStack Project Architecture Overview

by Ken Pepple

I thought it would be a good chance to revisit my earlier blog post on OpenStack Compute ("Nova") architecture. This time around, instead of detailing the architecture of just a single service, I'll look at all the pieces of the OpenStack project working together.

To level-set everyone's understanding, let's briefly review the OpenStack project components and history. Founded in 2010 by Rackspace and NASA, the project has

released four versions and is set to release the fifth ("Essex" or 2012.1) in April. Originally, it consisted of a trio of "core" services:

- Object Store ("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 my company Internap. In fact, the images for this blog post are being served via the Internap Swift implementation.

- Image ("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 ("Nova") provides virtual servers upon demand. Similar to Amazon's EC2 service, it also provides volume services analogous to Elastic Block Services (EBS). Internap provide a commercial compute service built on Nova and it is used internally at Mercado Libre and NASA (where it originated).

The upcoming release promotes two new projects to "core" project status:

- Dashboard ("Horizon") provides a modular web-based user interface for all the OpenStack services.

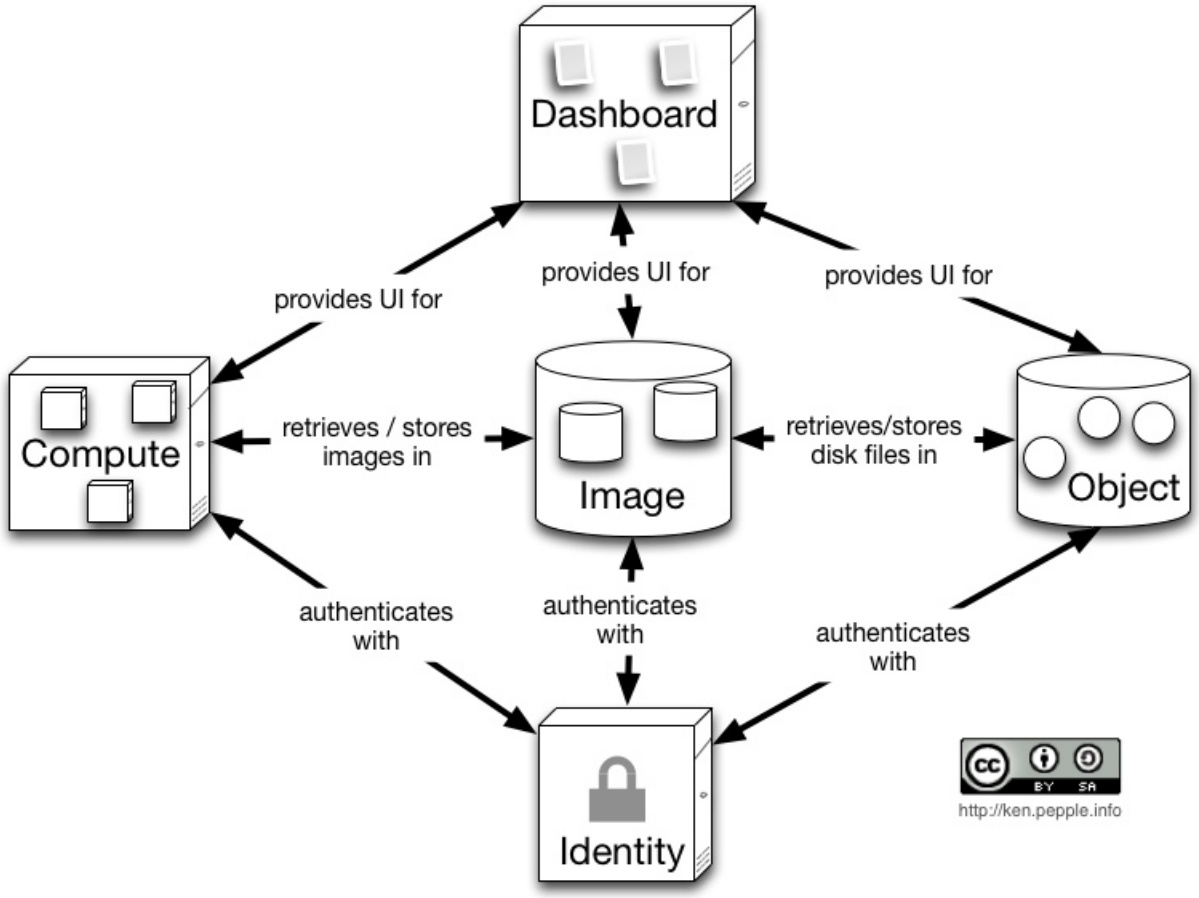
- Identity ("Keystone") provides authentication and authorization for all the OpenStack services. It also provides a service catalog of services within a particular deployment.

These new projects provide additional infrastructure to support the original three projects.

Conceptual Architecture

The OpenStack project as a whole is designed to "deliver(ing) 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:

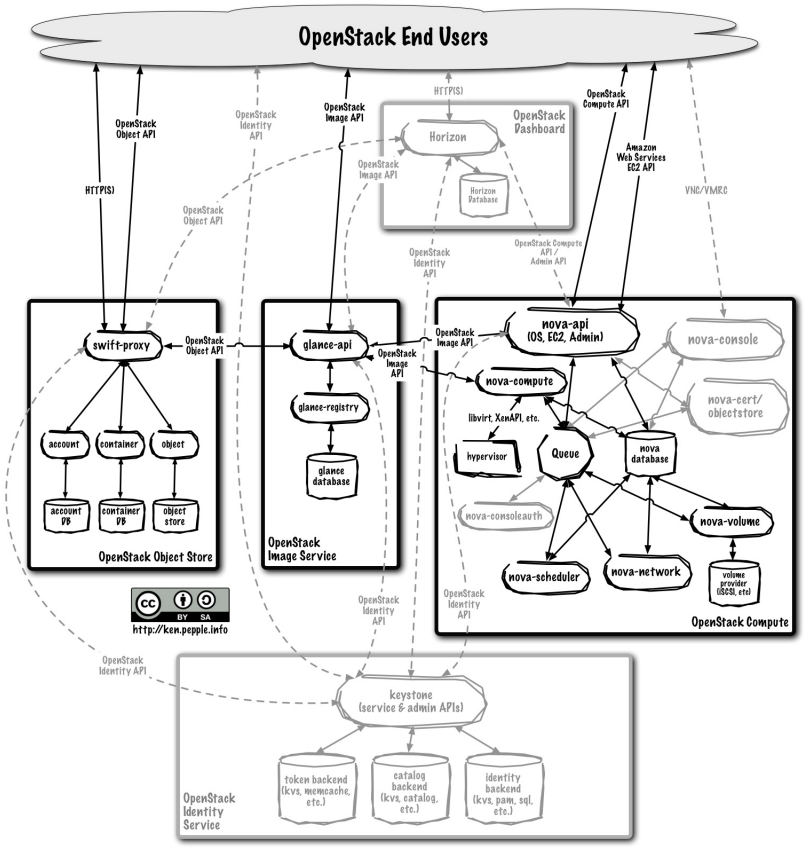


- Dashboard (Horizon) provides a web front end to the other OpenStack services
- Compute (Nova) stores and retrieves virtual disks ("images") and associated metadata in Glance
- Image (Glance) can store the actual virtual disk files in Object (Swift)
- All the services (*will eventually*) authenticate 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 compute users will use object storage heavily (and directly).

Logical Architecture

As you can imagine, the actual 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. In the diagram below, I illustrate what I believe will be the most common, "integrated" architecture of an OpenStack-based cloud.



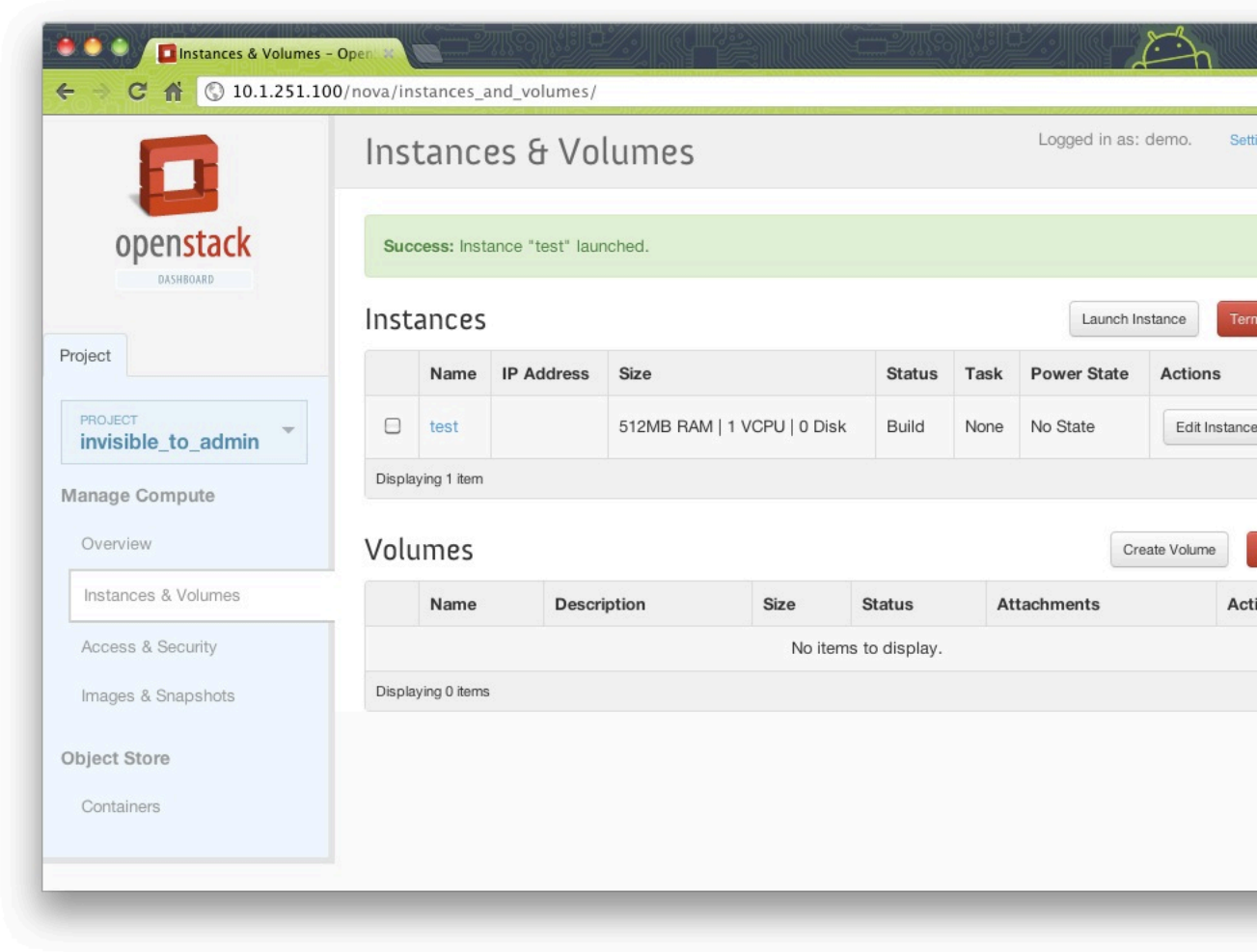
This picture is consistent with the description above in that:

- 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 seperated 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 to which one). As it relies mostly on the other services for data, it stores very little data of it's own.

From a network architecture point of view, this service will need to customer accessible as well as be able to talk to each services 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

Not much has really changed with Nova's architecture. They have added a few new helper services for EC2 compatibility and console services.

- `nova-api` accepts and responds to end user compute and volume API calls. It supports OpenStack 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 (mostly quota checks). In the Essex release, nova-api has been modularized, allowing for implementers to run only specific APIs.

- 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.

- `nova-volume` manages the creation, attaching and detaching of persistent volumes to compute instances (similar functionality to Amazon's Elastic Block Storage). It can use volumes from a variety of providers such as iSCSI or Rados Block Device in Ceph.

- The `nova-network` worker daemon is very similar to `nova-compute` and `nova-volume`. 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).

- 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 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).

- 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.

During the last two releases, Nova has augmented it's console services. Console services allow end users to access their virtual instance's console through a proxy. This involves a pair of new daemons (nova-console and nova-consoleauth).

Nova interacts with all of the usual suspects: Keystone for authentication, Glance for images and Horizon for web interface. The Glance interacts is interesting, though. 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 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 process 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

The Glance architecture has stayed relatively stable since the Cactus release. The biggest architectural change has been the addition of authentication, which was added in the Diablo release. Just as a quick reminder, 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, I have shown the most likely configuration (using Swift 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, 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 it's disk files in

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.

Future Projects

This completes the tour of the OpenStack Essex architecture. However, OpenStack will not be stopping here - the following OpenStack release ("Folsom") will welcome another core service to the fold:

- Network (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.

Although the release schedule for Folsom is not yet set (probably Fall 2012), I won't wait six months to update the picture for this.

Why Cloud?

In data centers today, many computers suffer the same underutilization 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.

2. Introduction to OpenStack Compute

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 ESX/ESXi 4.1 update 1
- Xen - Xen, Citrix XenServer and Xen Cloud Platform (XCP)

Users and Projects (Tenants)

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. With the use of the Identity Service, the issuing of a token also issues the roles assigned to the user, and the Identity Service calls projects tenants. Roles control the actions that a user is allowed to perform. For example, a user cannot allocate a public IP without the netadmin or admin role when the system is set up according to those rules. There are both global roles and per-project (tenant) role assignments. A user's access to particular images is limited by project, but the access key and secret key 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 project.

With the "–use_deprecated_auth" flag in place, OpenStack Compute uses a rights management system that employs a Role-Based Access Control (RBAC) model and supports the following five roles:

- Cloud Administrator (admin): Global role. Users of this class enjoy complete system access.
- IT Security (itsec): Global role. This role is limited to IT security personnel. It permits role holders to quarantine instances on any project.
- Project Manager (projectmanager): Project role. The default for project owners, this role affords users the ability to add other users to a project, interact with project images, and launch and terminate instances.
- Network Administrator (netadmin): Project role. Users with this role are permitted to allocate and assign publicly accessible IP addresses as well as create and modify firewall rules.
- Developer (developer): Project role. This is a general purpose role that is assigned to users by default.

While the original EC2 API supports users, OpenStack Compute adds the concept of projects, or tenants if your deployment uses the Identity Service (Keystone). Projects and tenants are isolated resource containers forming the principal organizational structure within Nova. They consist of a separate VLAN, volumes, instances, images, keys, and users. A user can specify which project or tenant he or she wishes to be known as by appending :project_id to his or her access key. If no project or tenant is specified in the API request, Compute attempts to use a project with the same id as the user.

For projects (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
- Publicly accessible IP addresses

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 nova-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 withing an OpenStack cloud to illustrate these concepts.

Initial State

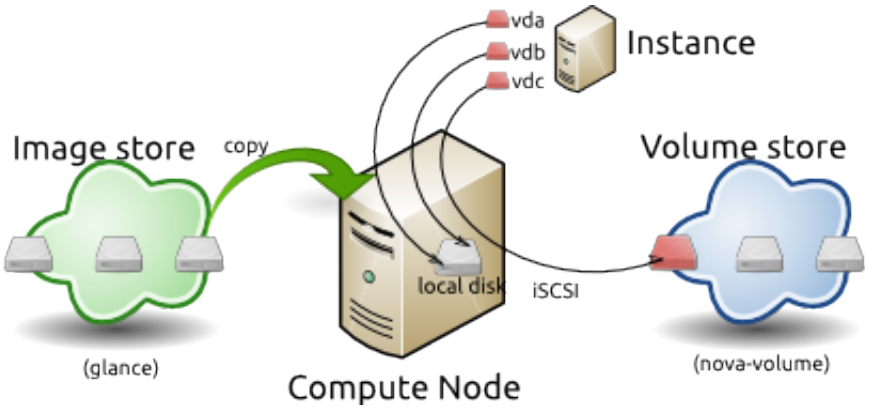
The following diagram shows the system state prior to launching an instance. The image store fronted by the image service, Glance, has some number of predefined images. In the cloud there is an available compute node with available vCPU, memory and local disk resources. Plus there are a number of predefined volumes in the nova-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 nova-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). The compute node attaches to the requested nova-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 dive. 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 goes away. 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 or 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 sill 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 it 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 out side of it, but are not features provided by the OpenStack software.

3. Installing OpenStack Compute

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 Service. You can install any of these projects separately and then configure them either as standalone or connected entities.

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 or SAS or SSD) Volume storage: two disks with 2 TB (SATA) for volumes attached to the compute nodes Network: one 1 GB 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. 32-bit processors will work for the cloud controller node. The package repositories referred to in this guide do not contain i386 packages.
Compute nodes (runs virtual instances)	Processor: 64-bit x86 Memory: 32 GB RAM Disk space: 30 GB (SATA) Network: two 1 GB NICs	Note that you cannot run 64-bit VM instances on a 32-bit compute node. A 64-bit compute node can run either 32- or 64-bit VMs, however. 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 XenServer installation guide and the XenServer harware compatibility list. For LXC, the VT extensions are not required. The packages referred to in this guide do not contain i386 packages.

Note

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

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

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. For Object Storage, the container and account servers use SQLite, and you can install it as part of the installation process.

Permissions: You can install OpenStack Compute, the Image Service, or Object Storage 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 keeps your cloud controller and compute nodes talking to the same time server to avoid problems 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.

Example Installation Architectures

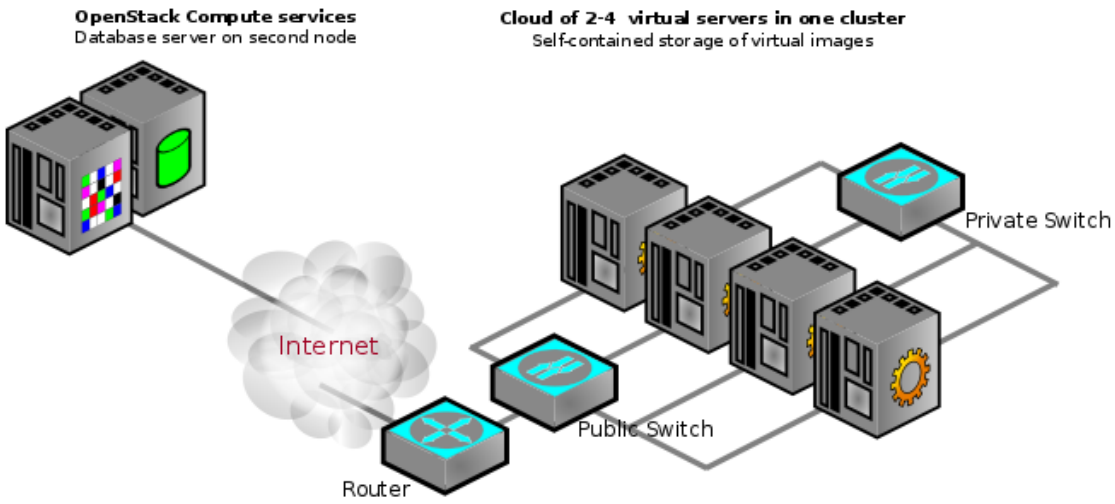
OpenStack Compute uses a shared-nothing, messaging-based architecture. While very flexible, the fact that you can install each nova- service on an independent server means there are many possible methods for installing OpenStack Compute. Here are the types of installation architectures:

- Single node: Only one server runs all nova- services and also drives all the virtual instances. Use this configuration only for trying out OpenStack Compute, or for development purposes.

- Two nodes: A cloud controller node runs the nova- services except for **nova-compute**, and a compute node runs **nova-compute**. A client computer is likely needed to bundle images and interfacing to the servers, but a client is not required. Use this configuration for proof of concepts or development environments.

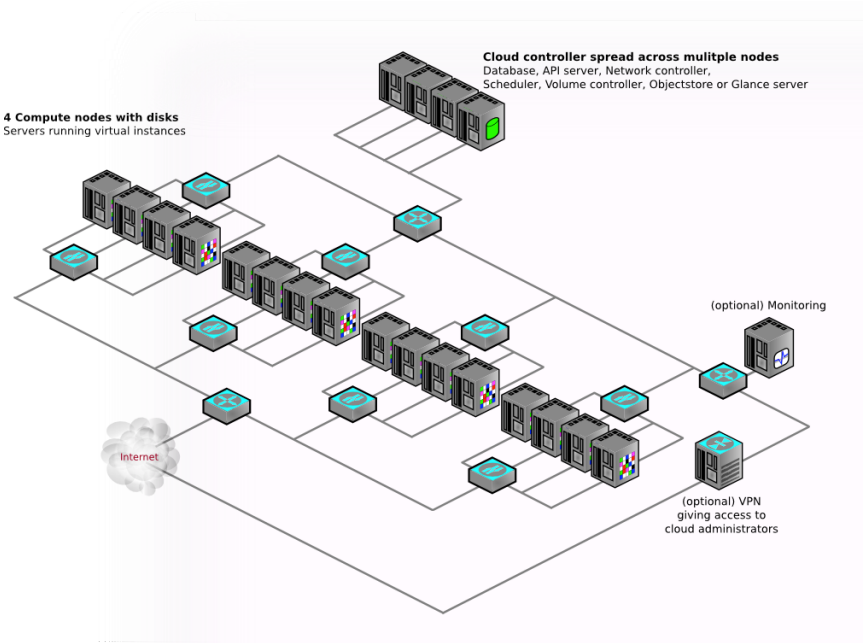
- Multiple nodes: You can add more compute nodes to the two node installation by simply installing **nova-compute** on an additional server and copying a `nova.conf` file to the added node. This would result in a multiple node installation. You can also add a volume controller and a network controller as additional nodes in a more complex multiple node installation. A minimum of 4 nodes is best for running multiple virtual instances that require a lot of processing power.

This is an illustration of one possible multiple server installation of OpenStack Compute; virtual server networking in the cluster may vary.



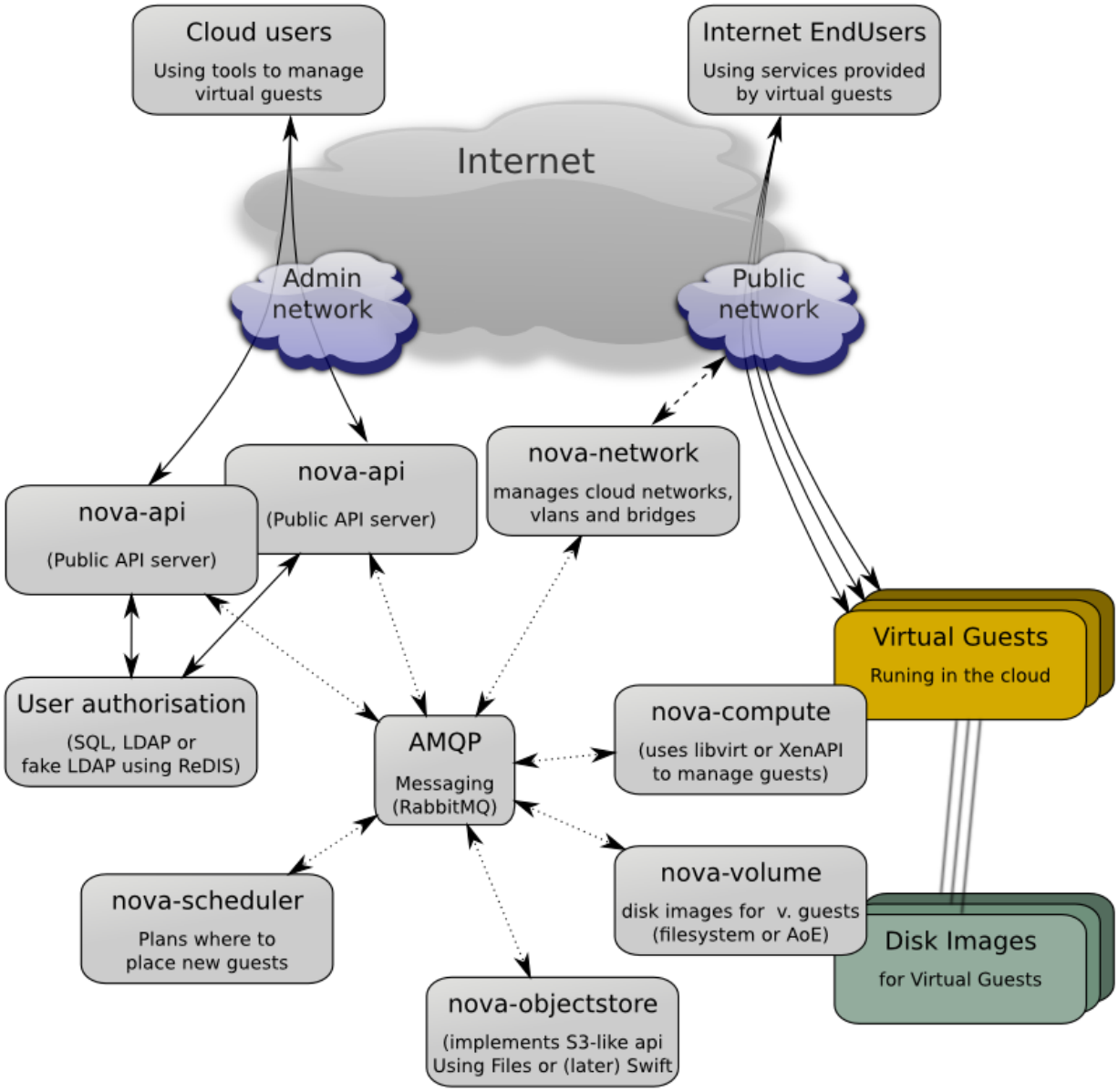
An alternative architecture would be to add more messaging servers if you notice a lot of back up in the messaging queue causing performance problems. In that case you would add an additional RabbitMQ server in addition to or instead of scaling up the database server. Your installation can run any nova- service on any server as long as the `nova.conf` is configured to point to the RabbitMQ server and the server can send messages to the server.

Multiple installation architectures are possible, here is another example illustration.



Service Architecture

Because Compute has multiple services and many configurations are possible, here is a diagram showing the overall service architecture and communication systems between the services.



Installing OpenStack Compute on Debian

Starting with Debian 7.0 "Wheezy", the OpenStack packages are provided as part of the distribution.

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

- `nova-api`
- `nova-scheduler`
- `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 Fedora or Red Hat Enterprise Linux 6

The Fedora project provides OpenStack packages in Fedora 16 and later. Fedora also provides packages for RHEL6 via the EPEL (Extra Packages for Enterprise Linux) 6 repository. If you would like to install OpenStack on RHEL6, see this page for more information on enabling the use of EPEL: http://fedoraproject.org/wiki/EPEL.

Detailed instructions for installing OpenStack Compute on Fedora or RHEL6 can be found on the Fedora wiki. See these pages for more information:

Getting Started with OpenStack on Fedora 17	The Essex release is in Fedora 17. This page discusses the installation of Essex on Fedora 17. Once EPEL 6 has been updated to include Essex, these instructions should be used if installing on RHEL 6. The main difference between the Fedora 17 instructions and what must be done on RHEL 6 is that RHEL 6 does not use systemd, so the **systemctl** commands will have to substituted with the RHEL 6 equivalent.
Getting Started with OpenStack Nova	This page was originally written as instructions for getting started with OpenStack on Fedora 16, which includes the Diablo release. At the time of writing, While EPEL 6 still includes Diablo, these instructions should be used if installing on RHEL 6.

Installing on Ubuntu

How you go about installing OpenStack Compute depends on your goals for the installation. You can use an ISO image, you can use a scripted installation, and you can manually install with a step-by-step installation.

ISO Distribution Installation

You can download and use an ISO image that is based on a Ubuntu Linux Server 10.04 LTS distribution containing only the components needed to run OpenStack Compute. See http://sourceforge.net/projects/stackops/files/ for download files and information, license information, and a README file. For documentation on the StackOps distro, see http://docs.stackops.org. For free support, go to http://getsatisfaction.com/stackops.

Scripted Installation

You can download a script for a standalone install for proof-of-concept, learning, or for development purposes for Ubuntu 11.04 at https://devstack.org.

1. Install Ubuntu 11.04 (Natty):

   In order to correctly install all the dependencies, we assume a specific version of Ubuntu to make it as easy as possible. OpenStack works on other flavors of Linux (and some folks even run it on Windows!) We recommend using a minimal install of Ubuntu server in a VM if this is your first time.

2. Download DevStack:

   ```
   $ git clone git://github.com/openstack-dev/devstack.git
   ```

   The devstack repo contains a script that installs OpenStack Compute, the Image Service and the Identity Service and offers templates for configuration files plus data scripts.

3. Start the install:

   ```
   $ cd devstack; ./stack.sh
   ```

   It takes a few minutes, we recommend reading the well-documented script while it is building to learn more about what is going on.

Manual Installation on Ubuntu

The manual installation involves installing from packages shipped on Ubuntu 12.04 as a user with root (or sudo) permission. The Oneiric OpenStack Starter Guide provides instructions for a manual installation using the packages shipped with Ubuntu 11.10. The OpenStack Install and Deploy Manual provides instructions for installing using Ubuntu 12.04 packages. Refer to those manuals for detailed instructions by going to http://docs.openstack.org and clicking the links next to the manual title.

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: "Installing XenServer and XCP".

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: "Xen, XenAPI, XenServer and XCP".

4. Configuring OpenStack Compute

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 Configuration 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

# AUTHENTICATION
auth_strategy=keystone

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
volume_group=nova-volumes
volume_name_template=volume-%08x
iscsi_helper=tgtadm
```

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

# COMPUTE
libvirt_type=qemu
connection_type=libvirt
instance_name_template=instance-%08x
api_paste_config=/etc/nova/api-paste.ini
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=br100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=10.0.0.0/24

# 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
```

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

```
$ sudo addgroup nova
```

The `nova.config` 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
```

You also need to populate the database with the network configuration information that Compute obtains from the `nova.conf` file.

```
$ nova-manage network create <network-label> <project-network> <number-of-
networks-in-project> <addresses-in-each-network>
```

Here is an example of what this looks like with real values entered:

```
$ nova-manage db sync
$ nova-manage network create novanet 192.168.0.0/24 1 256
```

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.

The nova-manage service 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). If this is not the case you will need to manually edit the sql db `networks` table.

When you run the **nova-manage network create** command, entries are made in the `networks` and `fixed_ips` tables. However, one of the networks listed in the `networks` table needs to be marked as bridge in order for the code to know that a bridge exists. The network in the Nova networks table is marked as bridged automatically for Flat Manager.

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
```

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 the **euca-authorize** command 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 |
 id | host     | binary         | topic     | report_count | disabled |
 availability_zone |
+---------------------+---------------------+------------+---------
+----+----------+----------------+-----------+--------------+----------
+-------------------+
| 2011-01-28 22:52:46 | 2011-02-03 06:55:48 | NULL       |       0 |  1 |
 osdemo02 | nova-network   | network   |        46064 |        0 | nova
       |
| 2011-01-28 22:52:48 | 2011-02-03 06:55:57 | NULL       |       0 |  2 |
 osdemo02 | nova-compute   | compute   |        46056 |        0 | nova
       |
| 2011-01-28 22:52:52 | 2011-02-03 06:55:50 | NULL       |       0 |  3 |
 osdemo02 | nova-scheduler | scheduler |        46065 |        0 | nova
       |
| 2011-01-29 23:49:29 | 2011-02-03 06:54:26 | NULL       |       0 |  4 |
 osdemo01 | nova-compute   | compute   |        37050 |        0 | nova
       |
| 2011-01-30 23:42:24 | 2011-02-03 06:55:44 | NULL       |       0 |  9 |
 osdemo04 | nova-compute   | compute   |        28484 |        0 | nova
       |
```

```
| 2011-01-30 21:27:28 | 2011-02-03 06:54:23 | NULL       |       0 |  8 |
 osdemo05 | nova-compute   | compute   |        29284 |        0 | nova
       |
+---------------------+---------------------+------------+---------
+----+----------+----------------+-----------+--------------+----------
+-------------------+
```

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:

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

Note

In the essex release, the **nova diagnostics** command is only supported with Xen-based hypervisors.

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.

Essex configuration using KVM, FlatDHCP, MySQL, Glance, LDAP, and optionally sheepdog, API is EC2

From gerrit.wikimedia.org, used with permission. Where you see parameters passed in, they are reading from Puppet configuration files. For example, a variable like <%= novaconfig["my_ip"] %> is for the puppet templates they use to deploy.

```
[DEFAULT]

verbose=True
auth_strategy=keystone
connection_type=libvirt
root_helper=sudo /usr/bin/nova-rootwrap
instance_name_template=i-%08x
daemonize=1
scheduler_driver=nova.scheduler.simple.SimpleScheduler
max_cores=200
my_ip=<%= novaconfig["my_ip"] %>
logdir=/var/log/nova
state_path=/var/lib/nova
lock_path=/var/lock/nova
sql_connection=mysql://<%= novaconfig["db_user"] %>:<%= novaconfig["db_pass"]
 %>@<%= novaconfig["db_host"] %>/<%= novaconfig["db_name"] %>
image_service=nova.image.glance.GlanceImageService
s3_host=<%= novaconfig["glance_host"] %>
glance_api_servers=<%= novaconfig["glance_host"] %>:9292
rabbit_host=<%= novaconfig["rabbit_host"] %>
cc_host=<%= novaconfig["cc_host"] %>
network_host=<%= novaconfig["network_host"] %>
ec2_url=http://<%= novaconfig["api_host"] %>:8773/services/Cloud
ec2_dmz_host=<%= novaconfig["api_ip"] %>
dmz_cidr=<%= novaconfig["dmz_cidr"] %>
libvirt_type=<%= novaconfig["libvirt_type"] %>
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
flat_network_dhcp_start=<%= novaconfig["dhcp_start"] %>
dhcp_domain=<%= novaconfig["dhcp_domain"] %>
network_manager=nova.network.manager.FlatDHCPManager
flat_interface=<%= novaconfig["network_flat_interface"] %>
flat_injected=False
flat_network_bridge=<%= novaconfig["flat_network_bridge"] %>
```

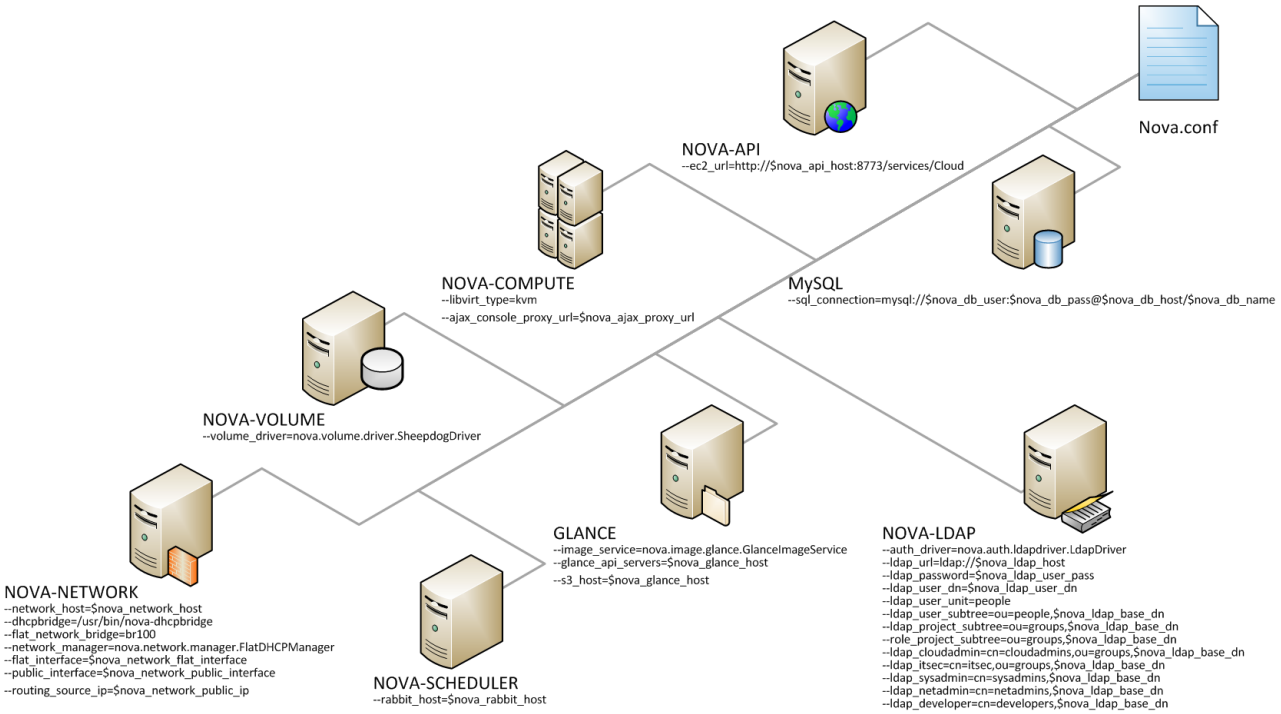
```
fixed_range=<%= novaconfig["fixed_range"] %>
public_interface=<%= novaconfig["network_public_interface"] %>
routing_source_ip=<%= novaconfig["network_public_ip"] %>
node_availability_zone=<%= novaconfig["zone"] %>
zone_name=<%= novaconfig["zone"] %>
quota_floating_ips=<%= novaconfig["quota_floating_ips"] %>
multi_host=True
api_paste_config=/etc/nova/api-paste.ini
#use_ipv6=True
allow_same_net_traffic=False
live_migration_uri=<%= novaconfig["live_migration_uri"] %>
```

These represent configuration role classes used by the puppet configuration files to build out the rest of the `nova.conf` file.

```
ldap_base_dn => "dc=wikimedia,dc=org",
ldap_user_dn => "uid=novaadmin,ou=people,dc=wikimedia,dc=org",
ldap_user_pass => $passwords::openstack::nova::nova_ldap_user_pass,
ldap_proxyagent => "cn=proxyagent,ou=profile,dc=wikimedia,dc=org",
ldap_proxyagent_pass =>
 $passwords::openstack::nova::nova_ldap_proxyagent_pass,
controller_mysql_root_pass =>
 $passwords::openstack::nova::controller_mysql_root_pass,
puppet_db_name => "puppet",
puppet_db_user => "puppet",
puppet_db_pass => $passwords::openstack::nova::nova_puppet_user_pass,
# By default, don't allow projects to allocate public IPs; this way we can
# let users have network admin rights, for firewall rules and such, and can
# give them public ips by increasing their quota
quota_floating_ips => "0",
libvirt_type => $realm ? {
   "production" => "kvm",
   "labs" => "qemu",
db_host => $controller_hostname,
  dhcp_domain => "pmtpa.wmflabs",
  glance_host => $controller_hostname,
  rabbit_host => $controller_hostname,
  cc_host => $controller_hostname,
  network_flat_interface => $realm ? {
   "production" => "eth1.103",
   "labs" => "eth0.103",
  },
  network_flat_interface_name => $realm ? {
   "production" => "eth1",
   "labs" => "eth0",
  },
  network_flat_interface_vlan => "103",
  flat_network_bridge => "br103",
  network_public_interface => "eth0",
  network_host => $realm ? {
   "production" => "10.4.0.1",
   "labs" => "127.0.0.1",
  },
  api_host => $realm ? {
   "production" => "virt2.pmtpa.wmnet",
   "labs" => "localhost",
  },
  api_ip => $realm ? {
```

```
 "production" => "10.4.0.1",
 "labs" => "127.0.0.1",
},
fixed_range => $realm ? {
 "production" => "10.4.0.0/24",
 "labs" => "192.168.0.0/24",
},
dhcp_start => $realm ? {
 "production" => "10.4.0.4",
 "labs" => "192.168.0.4",
},
network_public_ip => $realm ? {
 "production" => "208.80.153.192",
 "labs" => "127.0.0.1",
},
dmz_cidr => $realm ? {
 "production" => "208.80.153.0/22,10.0.0.0/8",
 "labs" => "10.4.0.0/24",
},
controller_hostname => $realm ? {
 "production" => "labsconsole.wikimedia.org",
 "labs" => $fqdn,
},
ajax_proxy_url => $realm ? {
 "production" => "http://labsconsole.wikimedia.org:8000",
 "labs" => "http://${hostname}.${domain}:8000",
},
ldap_host => $controller_hostname,
puppet_host => $controller_hostname,
puppet_db_host => $controller_hostname,
live_migration_uri => "qemu://%s.pmtpa.wmnet/system?pkipath=/var/lib/nova",
zone => "pmtpa",
keystone_admin_token => $keystoneconfig["admin_token"],
keystone_auth_host => $keystoneconfig["bind_ip"],
keystone_auth_protocol => $keystoneconfig["auth_protocol"],
keystone_auth_port => $keystoneconfig["auth_port"],
```

Figure 4.1. KVM, FlatDHCP, MySQL, Glance, LDAP, and optionally sheepdog



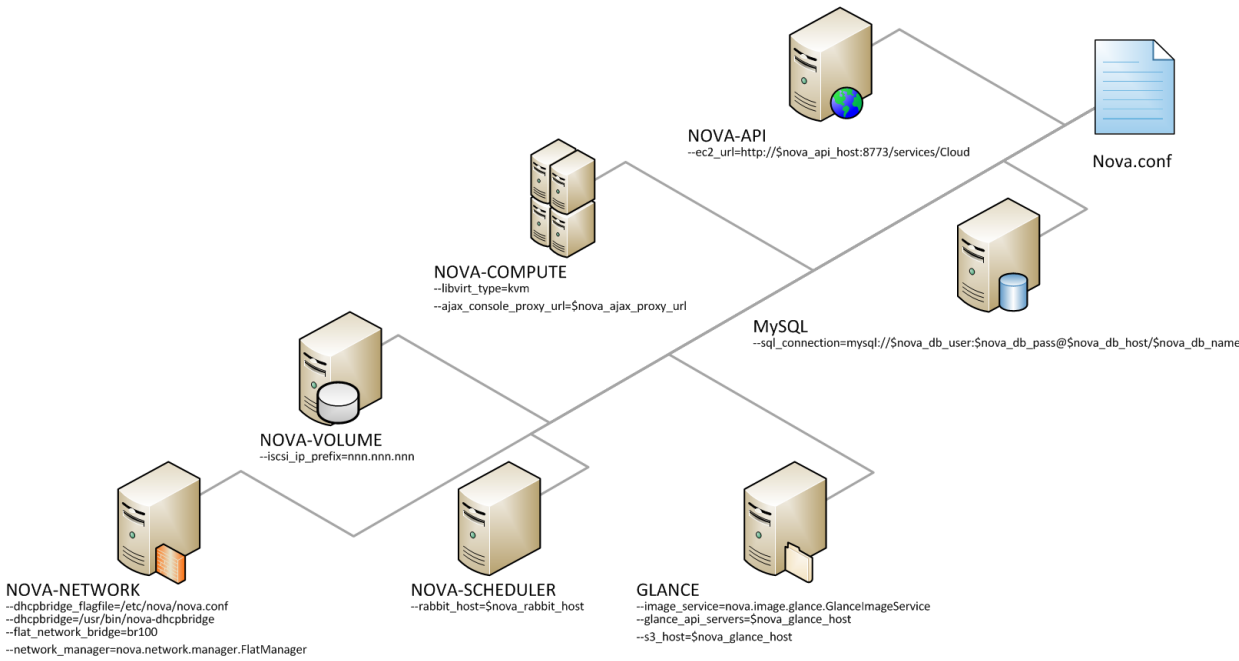
KVM, Flat, MySQL, and Glance, OpenStack or EC2 API

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

# AUTHENTICATION
auth_strategy=keystone

# SCHEDULER
compute_scheduler_driver=nova.scheduler.filter_scheduler.FilterScheduler

# VOLUMES
volume_group=nova-volumes
volume_name_template=volume-%08x
```

```
iscsi_helper=tgtadm

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

# COMPUTE
libvirt_type=qemu
connection_type=libvirt
instance_name_template=instance-%08x
api_paste_config=/etc/nova/api-paste.ini
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=br100
vlan_interface=eth0
flat_network_bridge=br100
flat_interface=eth0
fixed_range=10.0.0.0/24

# 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
```

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



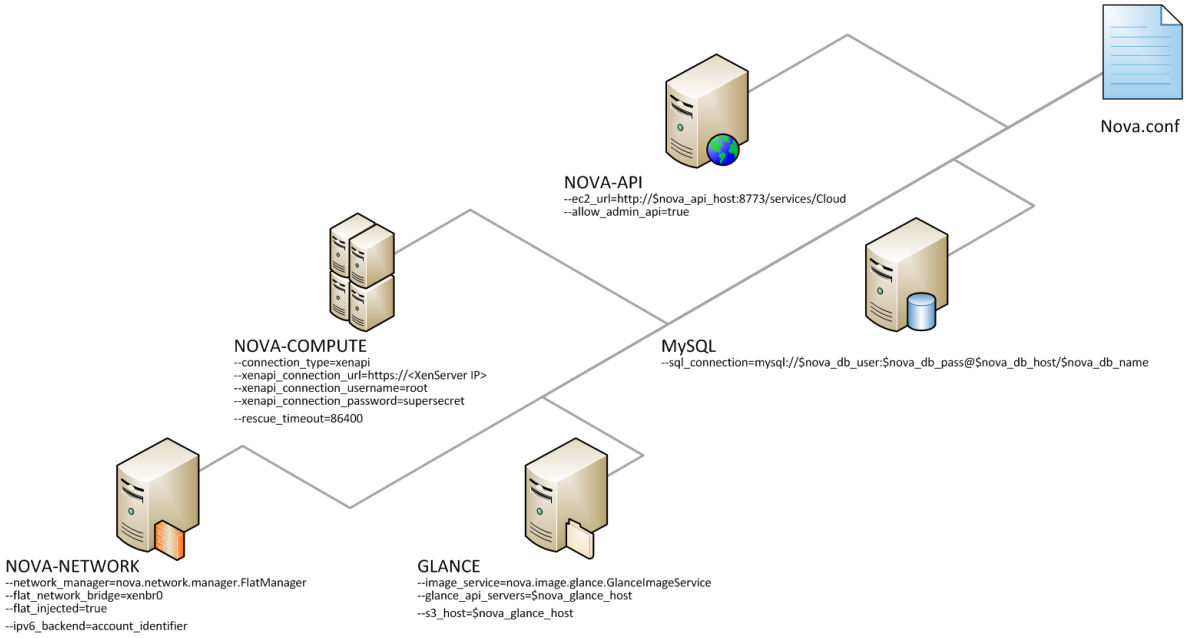
XenServer, Flat networking, MySQL, and Glance, OpenStack API

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
connection_type=xenapi
xenapi_connection_url=https://<XenServer IP>
xenapi_connection_username=root
xenapi_connection_password=supersecret
rescue_timeout=86400
allow_admin_api=true
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 4.3. 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

Configuration option=Default value	(Type) Description
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
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

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 the section called "Hypervisor Configuration Basics" [117] 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_PEER2PEER,VIR_MIGRATE_LIVE,VIR_MIGRATE_NON_SHARED_INC	(StrOpt) Migration flags to be set for block migration
checksum_base_images=false	(BoolOpt) Write a checksum for files in _base to disk
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_password=false	(BoolOpt) Inject the admin password at boot time, without an agent.
libvirt_nonblocking=false	(BoolOpt) Use a separated OS thread pool to realize non-blocking libvirt calls
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.LibvirtBridgeDriver	(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"	(StrOpt) Libvirt handlers 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 hard reboot if instance does not shutdown within this window.

Configuration option=Default value	(Type) Description
remove_unused_base_images=false	(BoolOpt) Should unused base images be removed?
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	(StrOpt) 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 connection_type is vmwareapi
vmwareapi_host_ip=<None>	(StrOpt) URL for connection to VMWare ESX host.Required if connection_type is vmwareapi.
vmwareapi_host_password=<None>	(StrOpt) Password for connection to VMWare ESX host. Used only if connection_type is vmwareapi.
vmwareapi_host_username=<None>	(StrOpt) Username for connection to VMWare ESX host. Used only if connection_type is vmwareapi.
vmwareapi_task_poll_interval=5.0	(FloatOpt) The interval used for polling of remote tasks. Used only if connection_type is vmwareapi
vmwareapi_vlan_interface=vmnic0	(StrOpt) Physical ethernet adapter name for vlan networking

Configuring Authentication and Authorization

There are different methods of authentication for the OpenStack Compute project, including no authentication, keystone, or deprecated (which uses nova-manage commands to create users). With additional configuration, you can use the OpenStack Identity Service, code-named Keystone. Refer to Chapter 6, *Identity Management* [75] 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, keystone, and deprecated.

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

Configuration option=Default value	(Type) Description
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
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, though floating IPs are not supported in the Bexar release. In VlanManager, different 64bit global routing prefix is used for each project. In FlatDHCPManager, one 64bit global routing prefix is used for all instances. The Cactus release includes support for the FlatManager networking mode with a required database migration.

This configuration has been tested on Ubuntu 10.04 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-manage network create** you can add a fixed range for IPv6 addresses. You must specify public or private after the create parameter.

```
$nova-manage network create public fixed_range num_networks network_size
 vlan_start vpn_start 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-manage network create public 10.0.1.0/24 3 32 100 1000 fd00:1::/48
```

Here is a usage example for FlatDHCPManager:

```
$ nova-manage network create public 10.0.2.0/24 3 32 0 0 fd00:1::/48
```

Note that `vlan_start` and `vpn_start` parameters are not used by FlatDHCPManager.

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

Configuration option=Default value	(Type) Description
fixed_range_v6=fd00::/48	(StrOpt) Fixed IPv6 address block

Configuration option=Default value	(Type) Description
gateway_v6=<None>	(StrOpt) Default IPv6 gateway
ipv6_backend=rfc2462	(StrOpt) Backend to use for IPv6 generation
use_ipv6=false	(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
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
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

Configuring Migrations

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 but the instances running on the host must be kept running. Migration can also be useful to redistribute the load when many VM instances are running on a specific physical machine.

The following section describes how to configure for migrations using the KVM hypervisor. The Xen hypervisor supports migrations as well, including live migration. See the XenServer Live Migration OpenStack wiki page for more details.

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

This feature for cloud administrators only, since the use of nova-manage is necessary.

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 Enabling true live migration 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. See important nova-compute options for more details on this option.

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, nova-volume, nova-objectstore, nova-scheduler.
- Host B and Host C 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 accross 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. 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.

3. 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
```

4. 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
```

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

```
$ HostA:/NOVA-INST-DIR/instances /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/sda1            921514972   4180880 870523828   1% /
none                  16498340      1228  16497112   1% /dev
none                  16502856         0  16502856   0% /dev/shm
none                  16502856       368  16502488   1% /var/run
none                  16502856         0  16502856   0% /var/lock
none                  16502856         0  16502856   0% /lib/init/rw
HostA:        921515008 101921792 772783104  12% /var/lib/nova/instances
 ( <--- this line is important.)
```

6. Update the libvirt configurations. Modify `/etc/libvirt/libvirt.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 succesfully restarted.

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

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

7. 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.

8. 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) Define live migration behavior
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER	(StrOpt) Define live migration behavior.
live_migration_retry_count=30	(IntOpt) Number of 1 second retries needed in live_migration
live_migration_uri=qemu+tcp://%s/system	(StrOpt) Define protocol used by live_migration feature

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 miration 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 migrated.

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 metadatas
- A chunk server (CSS) which store the datas as chunks and replicate them accross 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 :

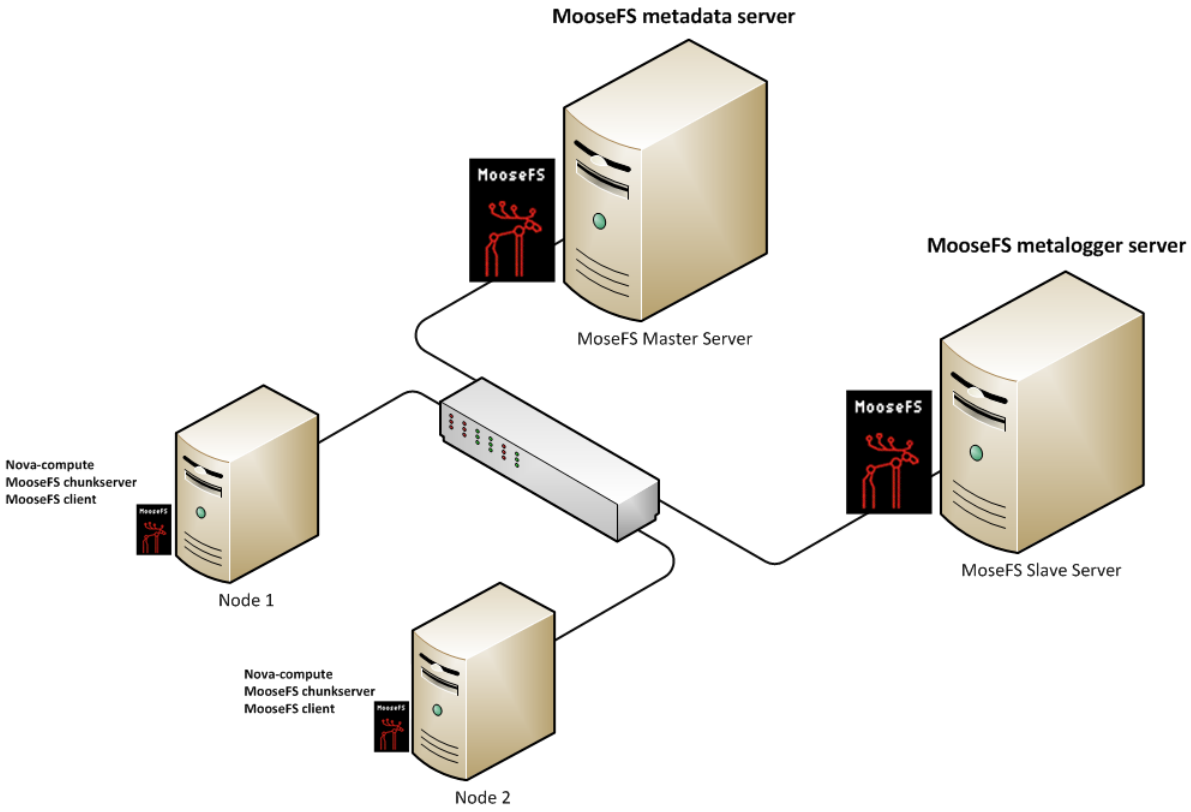
- Two compute nodes running both MooseFS chunkserver and client services.
- One MooseFS master server, running the metadata service.
- One MooseFS slave server, running the metalogger service.

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 4.4. 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 metadatas 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 the the MooseFS download page and fill the downlad form in order to obain 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 webinterface 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 cluser 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>
```

```
mfsappendchunks   mfschunkserver    mfsfileinfo       mfsgetgoal
 mfsmount          mfsrsetgoal       mfssetgoal        mfstools
mfscgiserv        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_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

Configuration option=Default value	(Type) Description
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

Configuring the Compute Messaging System

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

Configuration for RabbitMQ

OpenStack Compute 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.

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.

Configuration option	Default	Description
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 Compute. 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 Compute. It is not common that these options are used.

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

Configuration option	Default	Description
qpid_sasl_mechanisms	(Qpid default)	String value: A space separated list of acceptable SASL mechanisms to use for authentication.

Configuration option	Default	Description
qpid_reconnect_timeout	(Qpid default)	Integer value: The number of seconds to wait before deciding that a reconnect attempt has failed.
qpid_reconnect_limit	(Qpid default)	Integer value: The limit for the number of times to reconnect before considering the connection to be failed.
qpid_reconnect_interval_min	(Qpid default)	Integer value: Minimum number of seconds between connection attempts.
qpid_reconnect_interval_max	(Qpid default)	Integer value: Maximum number of seconds between connection attempts.
qpid_reconnect_interval	(Qpid default)	Integer value: Equivalent to setting qpid_reconnect_interval_min and qpid_reconnect_interval_max to the same value.
qpid_heartbeat	5	Integer value: Seconds between heartbeat messages sent to ensure that the connection is still alive.
qpid_tcp_nodelay	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 exchange to connect to
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 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
`enable_instance_password`	`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 a 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
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.

5. Reference for Configuration Options in nova.conf

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

Important

Nova options should *not* be quoted.

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.
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
aws_access_key_id=admin	(StrOpt) AWS Access ID
aws_secret_access_key=admin	(StrOpt) AWS Access Key
bandwith_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
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: []
connection_type=<None>	(StrOpt) Virtualization API connection type : libvirt, xenapi, or fake
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
control_exchange=nova	(StrOpt) the main RabbitMQ exchange to connect to
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 a 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
disable_process_locking=false	(BoolOpt) Whether to disable inter-process locks

Configuration option=Default value	(Type) Description
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)
enable_instance_password=true	(BoolOpt) Allows use of instance password during server creation
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
firewall_driver=nova.virt.firewall.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_servers=$glance_host:$glance_port	(ListOpt) A list of the glance API servers available to nova ([hostname|ip]:port)
glance_host=$my_ip	(StrOpt) default glance hostname or IP
glance_num_retries=0	(IntOpt) Number retries when downloading an image from glance
glance_port=9292	(IntOpt) default glance port
host=MGG2WEDRJM	(StrOpt) Name of this node. This can be an opaque identifier. It is not necessarily a hostname, FQDN, or IP address.
image_service=nova.image.glance.GlanceImageService	(StrOpt) The service to use for retrieving and searching images.
instance_dns_domain=	(StrOpt) full class name for the DNS Zone for instance IPs
instance_dns_manager=nova.network.dns_driver.DNSDriver	(StrOpt) full class name for the DNS Manager for instance IPs
instance_usage_audit_period=month	(StrOpt) time period to generate instance usages for. Time period must be hour, day, month or year
isolated_hosts=	(ListOpt) Host reserved for specific images
isolated_images=	(ListOpt) Images to run on isolated host
lock_path=$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

Configuration option=Default value	(Type) Description
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.notify_decorator,nova.compute.api:nova.notifier.api.notify_decorator	(ListOpt) List of modules/decorators to monkey patch
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
notification_driver=nova.notifier.no_op_notifier	(StrOpt) Default driver for sending 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.standard_extensions	(MultiStrOpt) Compute API 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.standard_extensions	(MultiStrOpt) osapi volume extension 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

Configuration option=Default value	(Type) Description
rabbit_use_ssl=false	(BoolOpt) connect over SSL for RabbitMQ
rabbit_userid=**guest**	(StrOpt) the RabbitMQ userid
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, *if* 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.
root_helper=sudo	(StrOpt) Command prefix to use for running commands as root
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_handler=nova.network.quantum.sg.NullSecurityGroupHandler	(StrOpt) The full class name of the security group handler class
service_down_time=60	(IntOpt) maximum time since last check-in for up service
start_guests_on_host_boot=false	(BoolOpt) Whether to (re-)start guests when the host reboots. If enabled, this option causes guests assigned to the host to be *unconditionally* restarted when nova-compute starts. If the guest is found to be stopped, it starts. If it is found to be running, it reboots.
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
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 cloudpipe 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_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

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
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

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
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

Configuration option=Default value	(Type) Description
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_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.
snapshot_name_template=snapshot-%08x	(StrOpt) Template string to be used to generate snapshot names
volume_name_template=volume-%08x	(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
quota_cores=20	(IntOpt) number of instance cores allowed per project (tenant)
quota_floating_ips=10	(IntOpt) number of floating ips allowed per project (tenant)
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

Configuration option=Default value	(Type) Description
quota_injected_file_path_bytes=255	(IntOpt) number of bytes allowed per injected file path
quota_injected_files=5	(IntOpt) number of injected files allowed
quota_instances=10	(IntOpt) number of instances allowed per project (tenant)
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)

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, keystone, and deprecated.
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

Configuration option=Default value	(Type) Description
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
ldap_user_subtree= "ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
ldap_user_unit= "Users"	(StrOpt) OID for Users

Configuration option=Default value	(Type) Description
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"

Configuration option=Default value	(Type) Description
vnc_enabled=true	(BoolOpt) enable VNC related features
vnc_keymap=en-us	(StrOpt) keymap for vnc
vncserver_listen=127.0.0.1	(StrOpt) IP address on which instance 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

Table 5.15. 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
dhcp_lease_time=120	(IntOpt) Lifetime of a DHCP lease in seconds
dhcpbridge=$bindir/nova-dhcpbridge	(StrOpt) location of nova-dhcpbridge
dhcpbridge_flagfile=/etc/nova/nova-dhcpbridge.conf	(StrOpt) location of flagfile for dhcpbridge
dmz_cidr=10.128.0.0/24	(StrOpt) dmz range that should be accepted
dns_server=<None>	(StrOpt) if set, uses specific dns server for dnsmasq
dnsmasq_config_file=	(StrOpt) Override the default dnsmasq settings with this file
linuxnet_interface_driver=nova.network.linux_net.LinuxBridgeInterfaceDriver	(StrOpt) Driver used to create ethernet devices.
linuxnet_ovs_integration_bridge=br-int	(StrOpt) Name of Open vSwitch bridge used with linuxnet
network_device_mtu=<None>	(StrOpt) MTU setting for vlan
networks_path=$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
auto_assign_floating_ip=false	(BoolOpt) Autoassigning floating IP to VM
cnt_vpn_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=novalocal	(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

Configuration option=Default value	(Type) Description
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
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
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

Table 5.16. Description of nova.conf file configuration options for live migration

Configuration option=Default value	(Type) Description
live_migration_bandwidth=0	(IntOpt) Define live migration behavior
live_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER	(StrOpt) Define live migration behavior.
live_migration_retry_count=30	(IntOpt) Number of 1 second retries needed in live_migration
live_migration_uri=qemu+tcp://%s/system	(StrOpt) Define protocol used by live_migration feature

Table 5.17. Description of nova.conf file configuration options for compute nodes

Configuration option=Default value	(Type) Description
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.

Configuration option=Default value	(Type) Description
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 5.18. Description of nova.conf file configuration options for bare metal deployment

Configuration option=Default value	(Type) Description
baremetal_driver=tilera	(StrOpt) Bare-metal driver runs on
tile_monitor=/usr/local/TileraMDE/bin/tile-monitor	(StrOpt) Tilera command line program for Bare-metal driver
baremetal_type=baremetal	(StrOpt) baremetal domain type
force_raw_images=true	(BoolOpt) Force backing images to raw format
img_handlers=loop,nbd,guestfs	(ListOpt) Order of methods used to mount disk images
injected_network_template=$pybasedir/nova/virt/interfaces.template	(StrOpt) Template file for injected network
max_nbd_devices=16	(IntOpt) maximum number of possible nbd devices
timeout_nbd=10	(IntOpt) time to wait for a NBD device coming up
virt_mkfs= "default=mkfs.ext3 -L %(fs_label)s -F %(target)s" virt_mkfs= "linux=mkfs.ext3 -L %(fs_label)s -F %(target)s" virt_mkfs= "windows=mkfs.ntfs –force –fast –label %(fs_label)s %(target)s"	(MultiStrOpt) mkfs commands for ephemeral device. The format is <os_type>=<mkfs command>

Table 5.19. Description of nova.conf file configuration options for hypervisors

Configuration option=Default value	(Type) Description
block_migration_flag=VIR_MIGRATE_UNDEFINE_SOURCE,VIR_MIGRATE_PEER2PEER,VIR_MIGRATE_LIVE,VIR_MIGRATE_NON_SHARED_INC	(StrOpt) Migration flags to be set for block migration
checksum_base_images=false	(BoolOpt) Write a checksum for files in _base to disk
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_password=false	(BoolOpt) Inject the admin password at boot time, without an agent.
libvirt_nonblocking=false	(BoolOpt) Use a separated OS thread pool to realize non-blocking libvirt calls
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.LibvirtBridgeDriver	(StrOpt) The libvirt VIF driver to configure the VIFs.
libvirt_volume_drivers="iscsi=nova.virt.libvirt.volume.LibvirtISCSIVolumeDriver,	(ListOpt) Libvirt handlers for remote volumes.

Configuration option=Default value	(Type) Description
local=nova.virt.libvirt.volume.LibvirtVolumeDriver, fake=nova.virt.libvirt.volume.LibvirtFakeVolumeDriver, rbd=nova.virt.libvirt.volume.LibvirtNetVolumeDriver,sheepdog=nova.virt.libvirt.volume.LibvirtNetVolumeDriver"	
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 hard reboot if instance does not shutdown within this window.
remove_unused_base_images=false	(BoolOpt) Should unused base images be removed?
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	(StrOpt) 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 connection_type is vmwareapi
vmwareapi_host_ip=<None>	(StrOpt) URL for connection to VMWare ESX host.Required if connection_type is vmwareapi.
vmwareapi_host_password=<None>	(StrOpt) Password for connection to VMWare ESX host. Used only if connection_type is vmwareapi.
vmwareapi_host_username=<None>	(StrOpt) Username for connection to VMWare ESX host. Used only if connection_type is vmwareapi.
vmwareapi_task_poll_interval=5.0	(FloatOpt) The interval used for polling of remote tasks. Used only if connection_type is vmwareapi
vmwareapi_vlan_interface=vmnic0	(StrOpt) Physical ethernet adapter name for vlan networking

Table 5.20. Description of nova.conf file configuration options for console access to VMs on VMWare VMRC or XenAPI

Configuration option=Default value	(Type) Description
console_driver=nova.console.xvp.XVPConsoleProxy	(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

Configuration option=Default value	(Type) Description
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

Table 5.21. Description of nova.conf file configuration options for S3 access to image storage

Configuration option=Default value	(Type) Description
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
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 5.22. 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
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.standard_filters	(MultiStrOpt) Filter classes available to the scheduler which may be specified more than once. An entry of "nova.scheduler.filters.standard_filters" maps to all filters included with nova.
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter	(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
least_cost_functions=nova.scheduler.least_cost.compute_fill_first_cost_fn	(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	(StrOpt) Driver to use for scheduling compute calls
volume_scheduler_driver=nova.scheduler.chance.ChanceScheduler	(StrOpt) 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

Configuration option=Default value	(Type) Description
max_networks=1000	(IntOpt) maximum number of networks to allow per host
skip_isolated_core_check=true	(BoolOpt) Allow overcommitting vcpus on isolated hosts

Table 5.23. Description of nova.conf file configuration options for volumes attached to VMs

Configuration option=Default value	(Type) Description
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
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

Configuration option=Default value	(Type) Description
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

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 request 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).

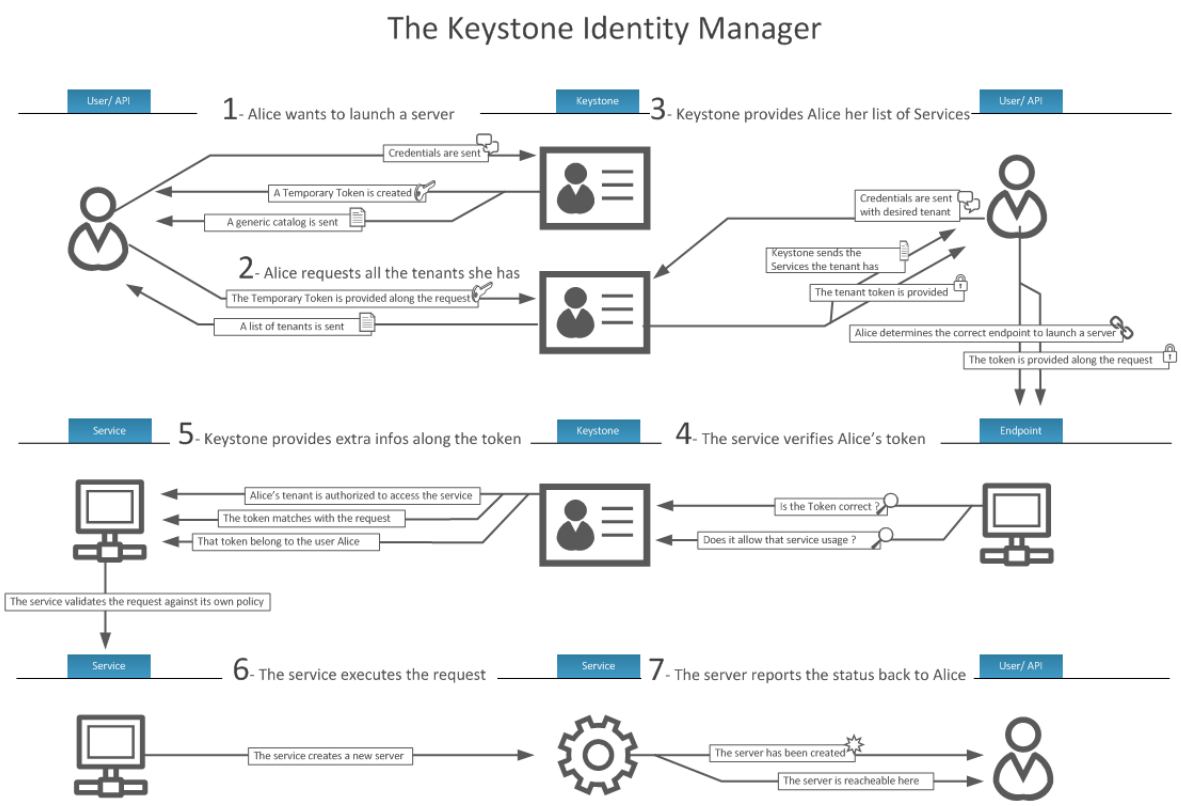
Examples are:

- 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 request 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 a 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 right 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 in instead of *tenant* in earlier versions of OpenStack Compute, some command-line tools use `--project_id` instead of `--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 tenant 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  |
+----------------------------------+---------+-------+--------+
| 96a6ebba0d4c441887aceaeced892585 |   True  |  ...  | alice  |
+----------------------------------+---------+-------+--------+
$ keystone role-list
+----------------------------------+--------------+
|                id                |     name     |
+----------------------------------+--------------+
| f8dd5a2e4dc64a41b96add562d9a764e | compute-user |
+----------------------------------+--------------+
$ keystone tenant-list
+----------------------------------+--------------+---------+
|                id                |     name     | enabled |
+----------------------------------+--------------+---------+
| 2395953419144b67955ac4bab96b8fd2 |     acme     |   True  |
+----------------------------------+--------------+---------+
$ keystone user-role-add  \
  --user=96a6ebba0d4c441887aceaeced892585  \
  --role=f8dd5a2e4dc64a41b96add562d9a764e  \
  --tenant_id=2395953419144b67955ac4bab96b8fd2
```

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: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.

Configuration File

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:

- `[identity]` - the python module that backends the identity system
- `[catalog]` - the python module that backends the service catalog
- `[token]` - the python module that backends the token providing mechanisms
- `[policy]` - the python module that drives the policy system for RBAC

The configuration file is expected to be named `keystone.conf`. When starting up 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 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.

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.

Migrating from legacy versions of keystone

Migration support is provided for the following legacy keystone versions:

- diablo-5
- stable/diablo
- essex-2
- essex-3

To migrate from legacy versions of Identity, use the following steps:

Step 1: Configure keystone.conf

It is important that the database that you specify be different from the one containing your existing install.

Step 2: db_sync your new, empty database

Run the following command to configure the most recent schema in your new Identity installation:

```
keystone-manage db_sync
```

Step 3: Import your legacy data

Use the following command to import your old data:

```
```

```
keystone-manage import_legacy [db_url, e.g. 'mysql://root@foobar/keystone']
```

Specify db_url as the connection string that was present in your old `keystone.conf` file.

Step 4: Import your legacy service catalog

While the older Identity service stored the service catalog in the database, the updated version configures the service catalog using a template file. An example service catalog template file may be found in etc/default_catalog.templates.

To import your legacy catalog, run this command:

```
keystone-manage export_legacy_catalog \
    [db_url e.g. 'mysql://root@foobar/keystone'] > \
    [path_to_templates e.g. 'etc/default_catalog.templates']
```

After executing this command, you will need to restart the keystone service to see your changes.

Migrating from Legacy Authentication

Migration of users, projects (aka tenants), roles and EC2 credentials is supported for the Diablo and Essex releases of Nova. To migrate your auth data from Compute, use the following steps:

Step 1: Export your data from Compute

Use the following command to export your data from Compute (nova):

```
nova-manage export auth > /path/to/dump
```

It is important to redirect the output to a file so it can be imported in a later step.

Step 2: db_sync your new, empty database

Run the following command to configure the most recent schema in your new installation:

```
keystone-manage db_sync
```

Step 3: Import your data to Keystone

To import your Compute auth data from a dump file created with **nova-manage**, run this command:

```
$ keystone-manage import_nova_auth [dump_file, e.g. /path/to/dump]
```

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_legacy`: Import a legacy (pre-essex) version of the db.
- `export_legacy_catalog`: Export service catalog from a legacy (pre-essex) db.
- `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 keystone endpoint to communicate with. For example, http://localhost:35357/v2.0'
- `--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 role add-user-role \
3a751f78ef4c412b827540b829e2d7dd \
```

```
03c84b51574841ba9a0d8db7882ac645 \
20601a7f1d94447daa4dff438cb1c209
```

remove-user-role

arguments

- role_id
- user_id
- tenant_id

example:

```
keystone remove-user-role \
19d1d3344873464d819c45f521ff9890 \
08741d8ed88242ca88d1f61484a0fe3b \
20601a7f1d94447daa4dff438cb1c209
```

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

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`.

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 admin service token for the service (from the Configuration step above) and include that as the key 'admin_token' in Nova's api-paste.ini.

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. In order to enable to S3 compatibility, add the following lines to the keystone.conf file :

   Add the following filter :

   ```
   [filter:s3_extension]
   paste.filter_factory = keystone.contrib.s3:S3Extension.factory
   ```

   And update the "admin_api" pipeline, by updating the following line :

```
[pipeline:admin_api]
pipeline = token_auth admin_token_auth xml_body json_body debug
 ec2_extension crud_extension admin_service
```

With :

```
[pipeline:admin_api]
pipeline = token_auth admin_token_auth xml_body json_body debug
 ec2_extension s3_extension crud_extension admin_service
```

3. Configure the paste file for swift-proxy (`/etc/swift/swift-proxy.conf`)

4. 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 authtoken keystone proxy-server

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

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

[filter:authtoken]
paste.filter_factory = keystone.middleware.auth_token:filter_factory
# Delaying the auth decision is required to support token-less
# usage for anonymous referrers ('.r:*').
delay_auth_decision = true
service_port = 5000
service_host = 127.0.0.1
auth_port = 35357
auth_host = 127.0.0.1
auth_token = ADMIN
admin_token = ADMIN

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

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

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

5. Restart swift services.

6. 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/swift-proxy.conf` to use S3token and Swift3 middleware.

   Here's an example:

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

[pipeline:main]
pipeline = catch_errors healthcheck cache swift3 s3token authtoken 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 = keystone.middleware.swift_auth:filter_factory
operator_roles = admin, swiftoperator

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

[filter:authtoken]
paste.filter_factory = keystone.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

It is possible to connect an LDAP backend with the Identity service Keystone.

1. **Setting up the LDAP backend**

   - Configuring Users

     The users will be stored into a collection

     ```
     ou=Users,$SUBTREEY
     ```

     that will make use of the standard LDAP objectClass

     ```
     inetOrgPerson
     ```

     (being defined in `/etc/openldap/schema/inetorgperson.ldiff`. You would only need two LDAP fields :**CN** and **CN**. The **CN** field will be used for the bind call, and is the **ID** field for the **user** object.

   - Configuring Tenants

     OpenStack tenants is also a collection. They are instances of the object **groupOfNames** (defined in `/etc/openldap/schema/core.ldiff`. In order to bind tenant to users, the user's **DN** should be indicated into the tenant's **members** attribute.

   - Configuring Roles

     Roles will be stored into the organizationalRole LDAP object class, into `/etc/openldap/schema/core.ldiff`. The assignment is indicated via the User's **DN** in the **roleOccupant** attribute.

2. **Setting up Keystone**

   - The "[LDAP]" stanza in the `keystone.conf` file allows you to specify the parameters related to the LDAP backend. Supported values are :

     - url
     - user
     - password

- suffix
- use_dumb_member
- user_tree_dn
- user_objectclass
- user_id_attribute
- tenant_tree_dn
- tenant_objectclass
- tenant_id_attribute
- tenant_member_attribute
- role_tree_dn
- role_objectclass
- role_id_attribute'
- role_member_attribute

Here is a typical set-up :

```
[ldap]
url = ldap://localhost
tree_dn = dc=exampledomain,dc=com
user_tree_dn = ou=Users,dc=exampledomain,dc=com
role_tree_dn = ou=Roles,dc=exampledomain,dc=com
tenant_tree_dn = ou=Groups,dc=exampledomain,dc=com
user = dc=Manager,dc=exampledomain,dc=com
password = freeipa4all
backend_entities = ['Tenant', 'User', 'UserRoleAssociation', 'Role']
suffix =cn=exampledomain,cn=com

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

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
cloudadmin= "cn=cloudadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Cloud Admins
developer= "cn=developers,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Developers
itsec= "cn=itsec,ou=Groups,dc=example,dc=com"	(StrOpt) CN for ItSec
netadmin= "cn=netadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for NetAdmins
password= "changeme"	(StrOpt) LDAP password

Configuration option=Default value	(Type) Description
suffix= "cn=example,cn=com"	(StrOpt) LDAP suffix
use_dumb_member=False	(BoolOpt) Simulates an LDAP member
project_subtree= "ou=Groups,dc=example,dc=com"	(StrOpt) OU for Projects
objectClass= inetOrgPerson	(StrOpt) LDAP objectClass to use
schema_version=2	(IntOpt) Current version of the LDAP schema
sysadmin= "cn=sysadmins,ou=Groups,dc=example,dc=com"	(StrOpt) CN for Sysadmins
url= "ldap://localhost"	(StrOpt) Point this at your ldap server
user= "dc=Manager,dc=example,dc=com"	(StrOpt) LDAP User
user_tree_dn="ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
user_dn= "cn=Manager,dc=example,dc=com"	(StrOpt) DN of Users
user_objectClass= inetOrgPerson	(StrOpt) DN of Users
user_id_attribute= cn	(StrOpt) Attribute to use as id
user_modify_only=false	(BoolOpt) Modify user attributes instead of creating/ deleting
user_name_attribute= cn	(StrOpt) Attribute to use as name
user_subtree= "ou=Users,dc=example,dc=com"	(StrOpt) OU for Users
user_unit= "Users"	(StrOpt) OID for Users
tenant_tree_dn="ou=Groups,dc=example,dc=com"	(StrOpt) OU for Tenants
tenant_objectclass= groupOfNames	(StrOpt) LDAP ObjectClass to use for Tenants
tenant_id_attribute= cn	(strOpt) Attribute to use as Tenant
tenant_member_attribute= member	(strOpt) Attribute to use as Member
role_tree_dn= "ou=Roles,dc=example,dc=com"	(strOpt) OU for Roles
role_objectclass= organizationalRole	(strOpt) LDAP ObjectClass to use for Roles
role_project_subtree= "ou=Groups,dc=example,dc=com"	(StrOpt) OU for Roles
role_member_attribute= roleOccupant	(StrOpt) Attribute to use as Role member
role_id_attribute= cn	(StrOpt) Attribute to use as Role

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 by 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 = keystone.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.

7. Image Management

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.

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 andare 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.

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.

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

hypverisors. 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 aa 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 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 wtih the metada service (e.g., retrieve ssh public key, execute user data script). There are several ways to implement this functionality, inclduign:

- Install a cloud-init RPM , 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

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 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/nbd0*
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 add 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 add 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-x86_64
```

```
$ image=cirros-0.3.0-x86_64-disk.img
$ glance add 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 login 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 in to 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

vCPUS, 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 node to boot instances on

It is possible to specify which node to run the instance on using the nova client. In order to use such feature, make sure you are using an admin accont ; also, in your `/etc/nova/nova.conf`, make sure the following configuration option is set :

```
allow_admin_api
```

You can retrieve the current active node by running :

```
$nova-manage service list
```

```
server1 nova-network enabled  :- ) 2011-04-06 17:05:11
server1 nova-compute enabled  :- ) 2011-04-06 17:05:13
server1 nova-scheduler enabled :- ) 2011-04-06 17:05:17
server2 nova-compute disabled  :- ) 2011-04-06 17:05:19
```

We see here our "server2" runs as a node. You can now select the host on which the instance would be spawned, using the "–hint" flag :

```
$ nova boot --image 1 --flavor 2 --key_name test --hint force_hosts=server2
 my-first-server
```

Image management

CSS Corp- Open Source Services

by CSS Corp Open Source Services

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/cactus/openstack-compute/admin/content/starting-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 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
```

Shutdown 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 a 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 value 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 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  /               ext4    errors=
remount-ro  0       1
```

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 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 add 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. Sso 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
 -boot d 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 add 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 pacakge from qemu to qemu
  ii  qemu-common                    0.14.0~rc1+noroms-0ubuntu4~ppalucid1
      qemu common functionality (bios, documentati
  ii  qemu-kvm                       0.14.0~rc1+noroms-0ubuntu4~ppalucid1
      Full virtualization on i386 and amd64 hardwa
  ```

  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 (via nova-volume) to the instance. Thus, you should dettach 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.

8. Hypervisors

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. You cannot configure more than one virtualization type on the compute nodes, so the hypervisor selection is for the entire installation. These 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 ESX/ESXi 4.1 update 1, runs VMWare-based Linux and Windows images through a connection with the ESX server.
- 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.

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 8.1. Description of nova.conf configuration options for the compute node

Configuration Option	Default	Description
connection_type	default: 'libvirt'	libvirt, xenapi, or fake; Value that indicates the virtualization connection type

Configuration Option	Default	Description
compute_manager	default: 'nova.compute.manager.ComputeManager'	String value; Manager to use for nova-compute
compute_driver	default: 'nova.virt.connection.get_connection'	String value; Driver to use for controlling virtualization
images_path	default: '$state_path/images'	Directory; Location where decrypted images are stored on disk (when not using Glance)
instances_path	default: '$state_path/instances'	Directory; Location where instances are stored on disk (when not using Glance)
libvirt_type	default: 'kvm'	String; Libvirt domain type (valid options are: kvm, qemu, uml, xen)
allow_project_net_traffic	default: 'true'	true or false; Indicates whether to allow in-project network traffic
firewall_driver	default: 'nova.virt.libvirt_conn.IptablesFirewallDriver'	String; Firewall driver for instances, defaults to iptables
injected_network_template	default: ''	Directory and file name; Template file for injected network information
libvirt_uri	default: empty string	String; Override the default libvirt URI (which is dependent on libvirt_type)
libvirt_xml_template	default: ''	Directory and file name; Libvirt XML template
libvirt_inject_password	default: 'false'	When set, libvirt will inject the admin password into instances before startup. An agent is not required in the instance. The admin password is specified as part of the server create API call. If no password is specified, then a randomly generated password is used.
use_cow_images	default: 'true'	true or false; Indicates whether to use copy-on-write (qcow2) images. If set to false and using qemu or kvm, backing files will not be used.
force_raw_images	default: 'true'	true or false; If true, backing image files will be converted to raw image format.
rescue_image_id	default: 'ami-rescue'	String; AMI image to use for rescue
rescue_kernel_id	default: 'aki-rescue'	String; AKI image to use for rescue
rescue_ramdisk_id	default: 'ari-rescue'	String; ARI image to use for rescue
libvirt_nonblocking	default: 'false'	When set to 'true', libvirt APIs will be called in a seperate OS thread pool to avoid blocking the main thread. This feature is especially desirable if you use the snapshot feature, which has a notably long execution time, or have many instances in a given compute node. The feature is experimental and is disabled by default.

KVM

KVM is configured as the default hypervisor for Compute. To enable KVM explicitly, add the following configuration options `/etc/nova/nova.conf`:

```
connection_type=libvirt
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 Virtualizaiton Host Configuration and Guest Installation Guide.
- openSUSE: Installing KVM from the openSUSE Virtualization with KVM manual.
- SLES: Installing KVM from the SUSE Linux Enterprise ServerVirtualization 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, 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:

```
# 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
```

Troubleshooting

Trying to launch a new virtual machine instance fails with the ERR0R 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.

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.

running on

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:

```
connection_type=libvirt
libvirt_type=qemu
```

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) 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`:

```
connection_type=xenapi
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 as of the Essex release. To experiment using Xen through libvirt add the following configuration options `/etc/nova/nova.conf`:

```
connection_type=libvirt
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 toolstack you are using. Make sure you know what toolstack you want before you get started.

Xen Cloud Platform (XCP) is an open source (GPLv2) toolstack 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 avalible 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 toolstack and managment 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 provide 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 specifc 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 sofware (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 likly 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.

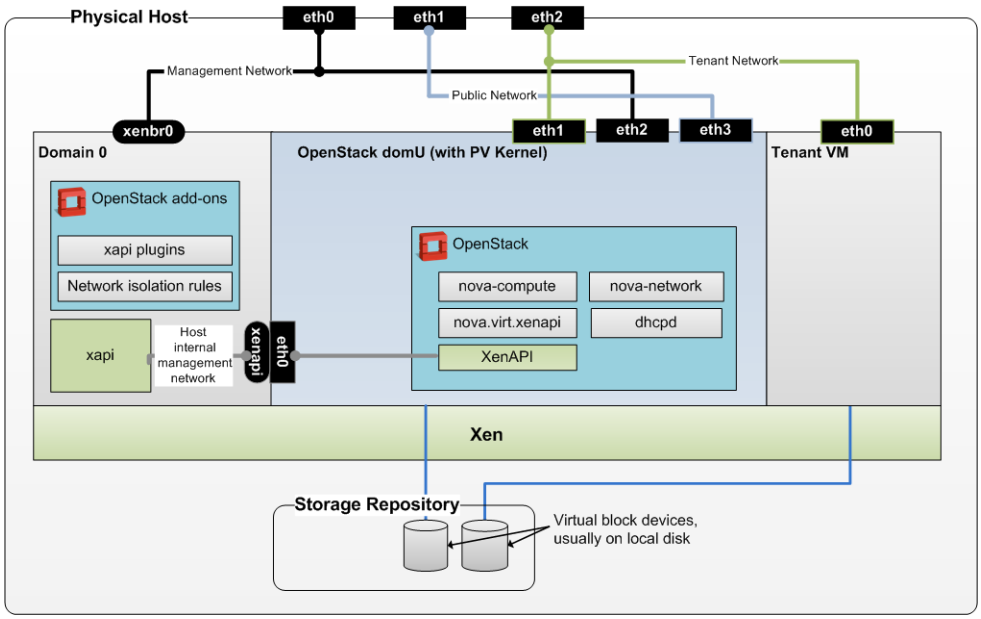
Paravirtualized versus hardware virtualized domains

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 domU: 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 Host Internal 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 traffic (RabbitMQ, MySQL, etc), Tenant network traffic (controlled by nova-network) and Public traffic (floating IPs, public API end points).
- Each network that leaves the host has been put through a separate physical network interface. This is the simplest model, but it's not the only one possible. You may choose to isolate this traffic using VLANs instead, for example.

XenAPI pools

Before OpenStack 2012.1 ("Essex"), all XenServer machines used with OpenStack are standalone machines, usually only using local storage.

However in 2012.1 and later, the host-aggregates feature allows you to create pools of XenServer hosts (configuring shared storage is still an out of band activity). This move will 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 supplimental 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 exsiting OS on top of Xen as Dom0. It is in Dom0 that the xapi deamon will run. You can find more details on the Xen.org wiki: http://wiki.xen.org/wiki/Project_Kronos

Post install steps

Now you have installed XenServer or XCP, it is time to start running OpenStack. Before you can start running OpenStack you must ensure:

- 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 repositry (SR) is a XenAPI specific term relating to the physical storage on which virtual disks are stored.
- Enable passwordless SSH login between all your XenServer hosts if you want to use the resize or migration functionality.
- Create the directory "/images" if you want resize or migration to work.

You are now ready to install OpenStack onto your XenServer system. This process involves the following steps:

- Install the VIF isolation rules to help prevent mac and ip address spoofing.
- Install the XenAPI plugins.
- 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 how to perform 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.

Further reading

Here are some of the resources avaiable 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.

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.

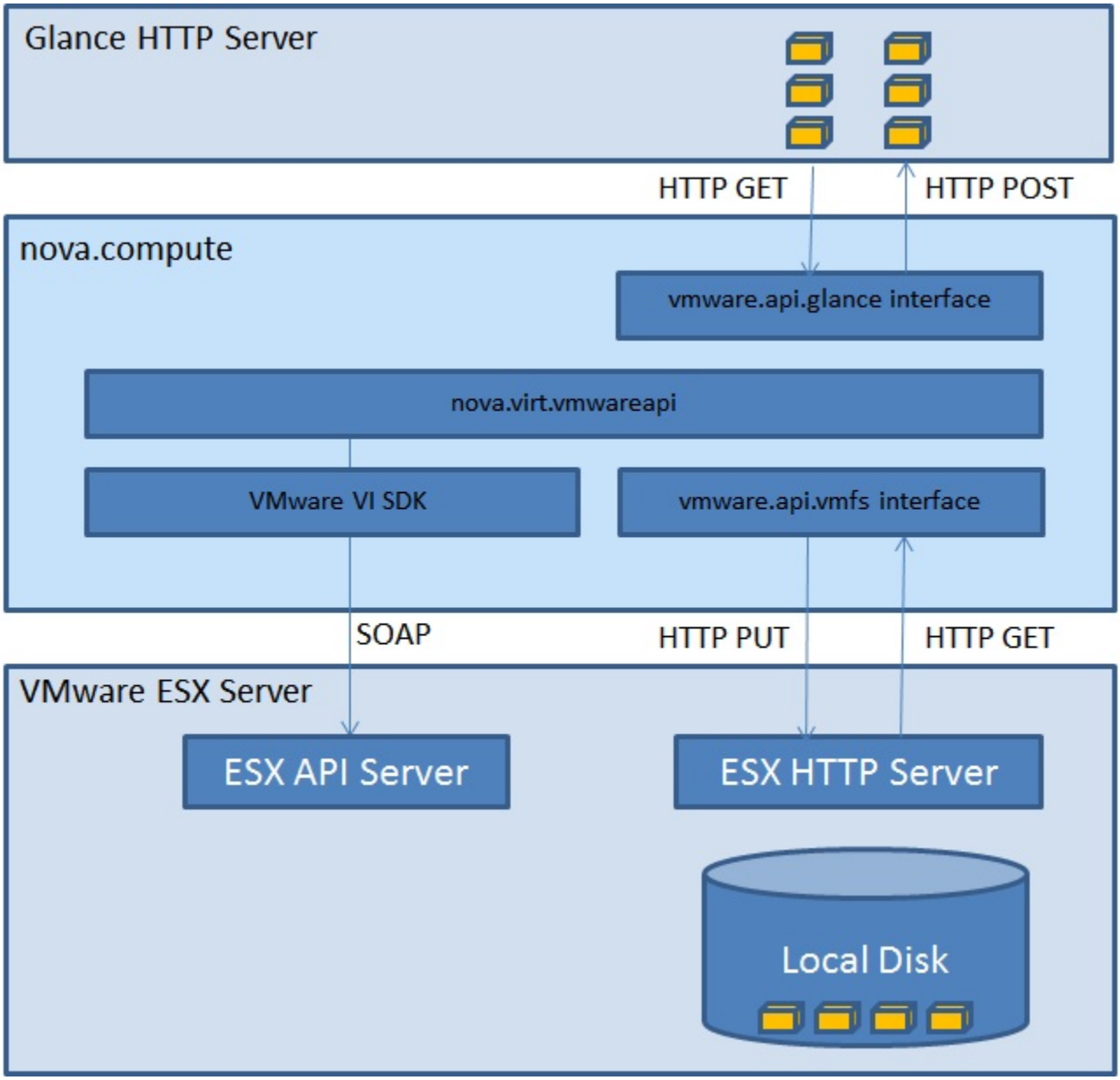
```
connection_type=libvirt
libvirt_type=lxc
```

On Ubuntu 12.04, enable LXC support in OpenStack by installing the `nova-compute-lxc` package.

VMware ESX/ESXi Server Support

Introduction

OpenStack Compute supports the VMware ESX hypervisor. This section describes the additional configuration required to launch VMWare-based virtual machine images.



Prerequisites

You will need to install the following software:

- python-suds: This software is needed by the nova-compute service. If not installed, the "nova-compute" service shuts down with the message: "Unable to import suds".
- SSH server
- Tomcat server

On ubuntu, these packages can be installed by doing (as root):

```
# apt-get install python-suds openssh-server tomcat6
```

Configure Tomcat to serve WSDL files

Download the WSDLs from http://www.vmware.com/support/developer/vc-sdk/ and copy the `wsdl` folder into `/var/lib/tomcat6/webapps`.

VMWare configuration options

Configure nova.conf with the following VMWare-specific config options:

```
vmwareapi_host_ip=<ESX hypervisor machine IP>
vmwareapi_host_username=< ESX hypervisor username>
vmwareapi_host_password=< ESX hypervisor password>
vmwareapi_wsdl_loc=http://127.0.0.1:8080/wsdl/vim25/vimService.wsdl
```

9. Networking

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.

In Compute, users organize their cloud resources in projects. A Compute project consists of a number of VM instances created by a user. For each VM instance, Compute assigns to it a private IP address. (Currently, Compute only supports Linux bridge networking that allows the virtual interfaces to connect to the outside network through the physical interface.)

The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network.

Currently, Compute 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 create or configure any network interfaces (except bridges 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 network administrator must configure the Linux networking bridge (typically named `br100`, although this configurable) both on the network controller hosting the network and on the cloud controllers hosting the instances. 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 configured with the `fixed_range` configuration option in `nova.conf`. 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

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, replacing the flat_network_dhcp_start configuration option that was available in Diablo.

Metadata service

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`. For example, instances 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/latest/meta-data/public-keys/0/openssh-key
```

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/latest/user-data
```

The Compute metadata service is compatible with the Amazon EC2 metadata service; virtual machine images that are designed for EC2 will work properly with OpenStack.

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-manage network create label CIDR n n` command to create the subnet that the VMs reside on.
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.

When you issue the `nova-manage network create` command, it uses the settings from the nova.conf configuration options file. Use the following command to create the subnet that your VMs will run on :

```
nova-manage network create private 192.168.0.0/24 1 256
```

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_injection` 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 configured with the `--fixed_range=` directive in nova.conf and the `--flat_network_bridge` setting. 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.

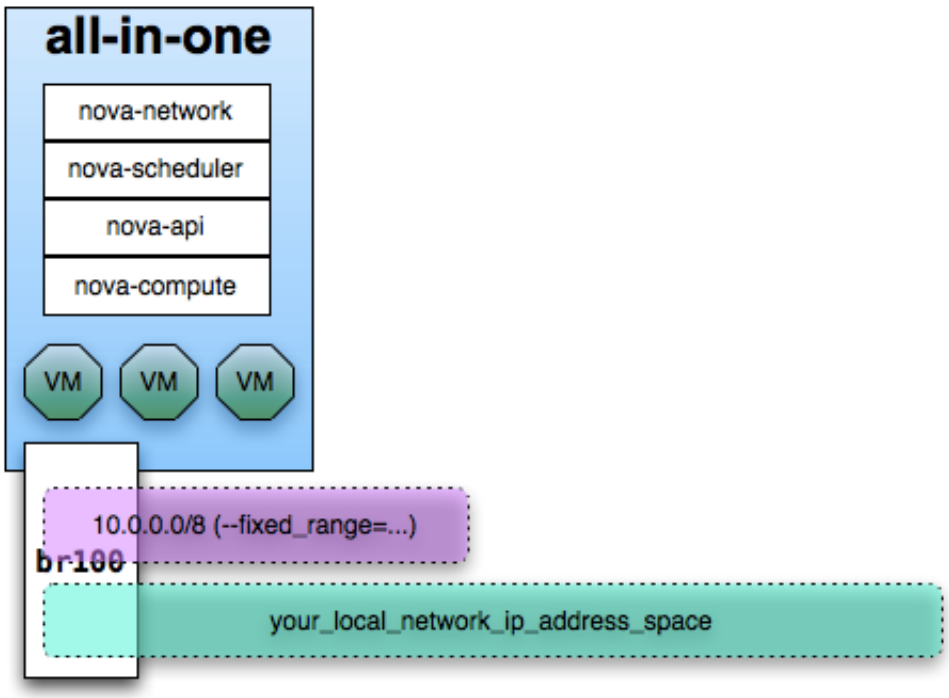
Set the compute node's external IP address to be on the bridge and add eth0 to that bridge. To do this, edit your network interfaces configuration to look like the following example:

```
# 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`

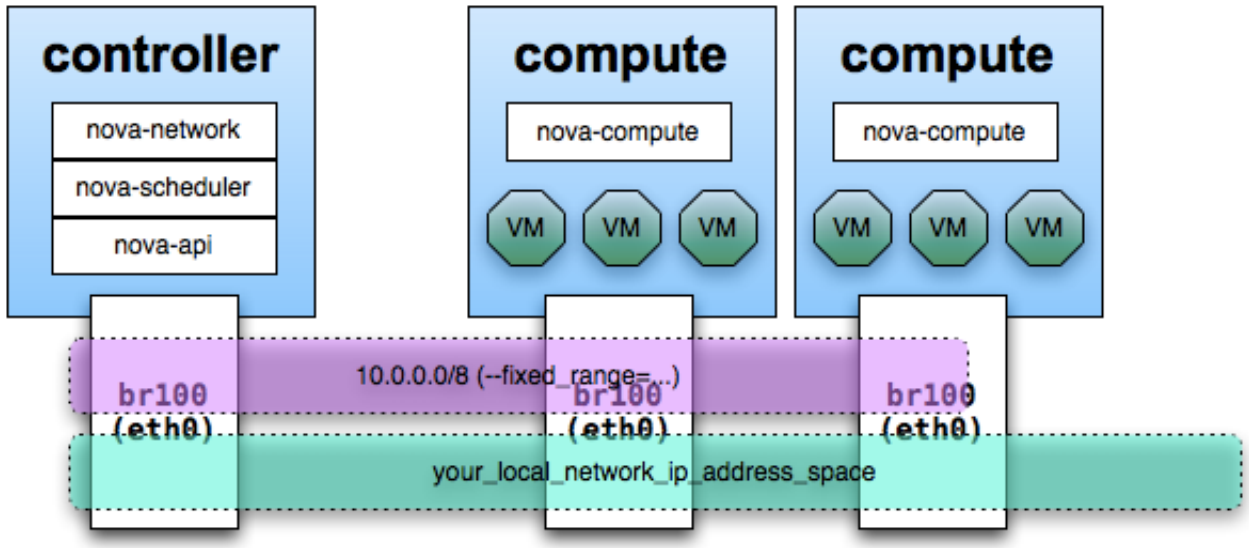
For an all-in-one development setup, this diagram represents the network setup.

Figure 9.1. Flat network, all-in-one server installation



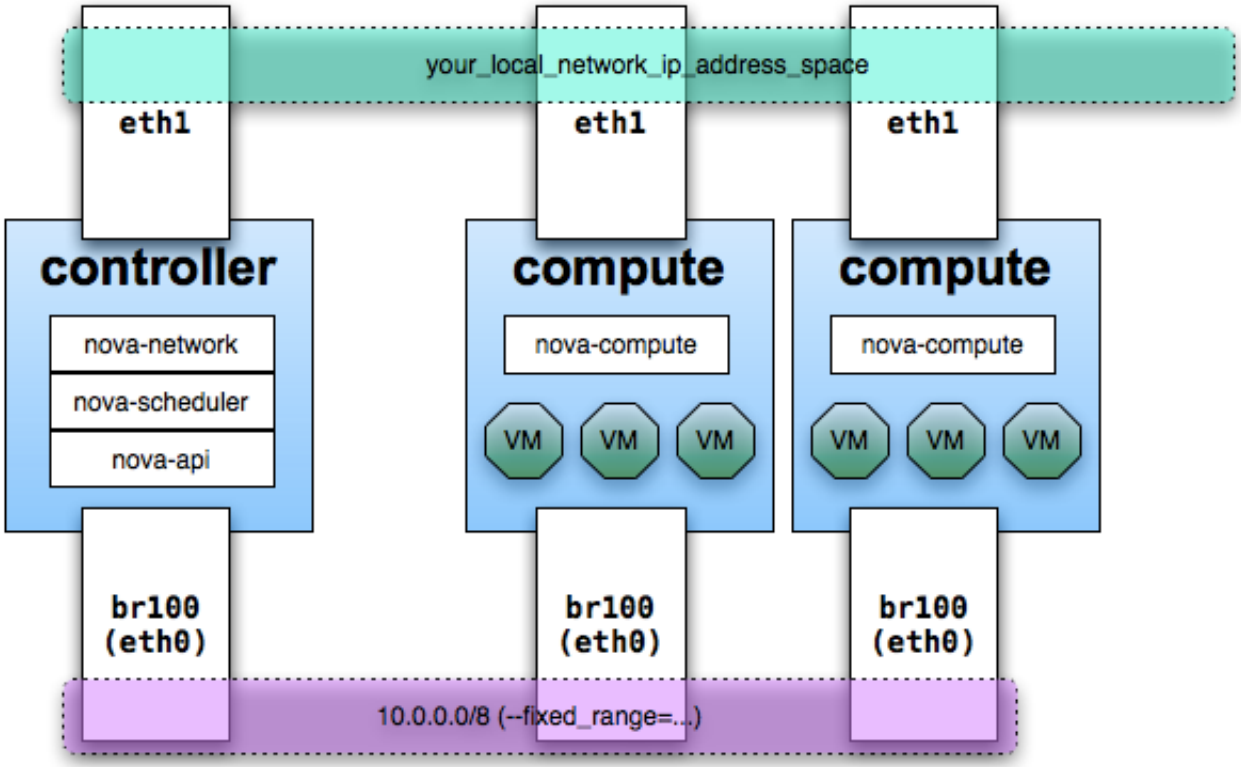
For multiple compute nodes with a single network adapter, which you can use for smoke testing or a proof of concept, this diagram represents the network setup.

Figure 9.2. 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 9.3. Flat network, multiple interfaces, multiple servers



XenAPI Flat Networking

When using the XenAPI driver, the virtual machines creates 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).

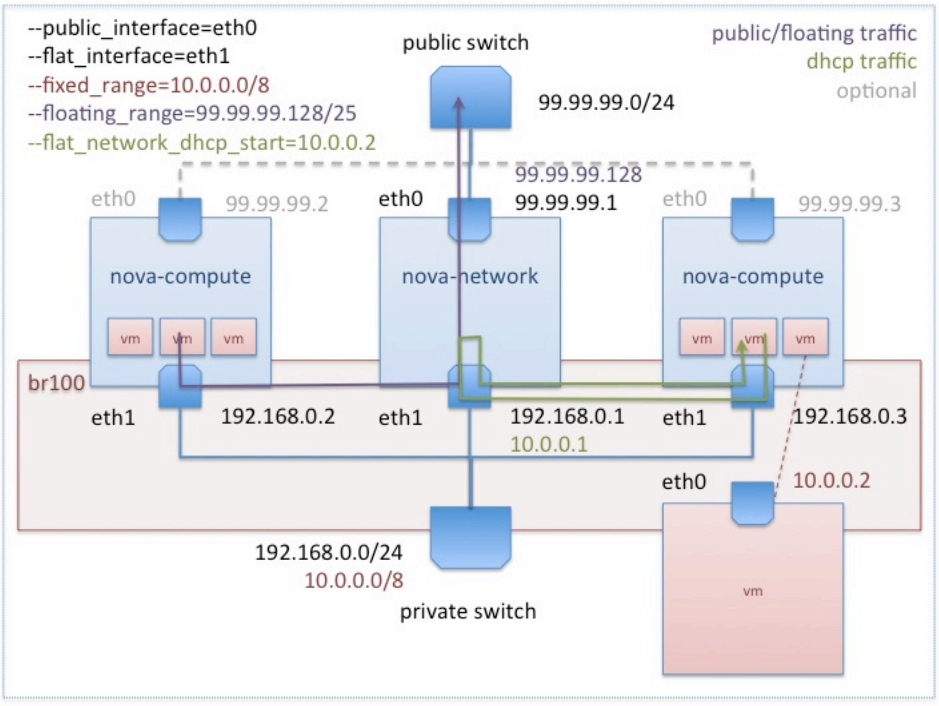
You can read a detailed walk-through of what exactly happens in single-host Flat DHCP mode in this blogpost, parts of which are also relevant in other networking modes.

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 9.4. 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 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. You can also add the interface to br100 manually and not set flat_interface. If this is the case for you, edit your nova.conf file to contain the following lines:

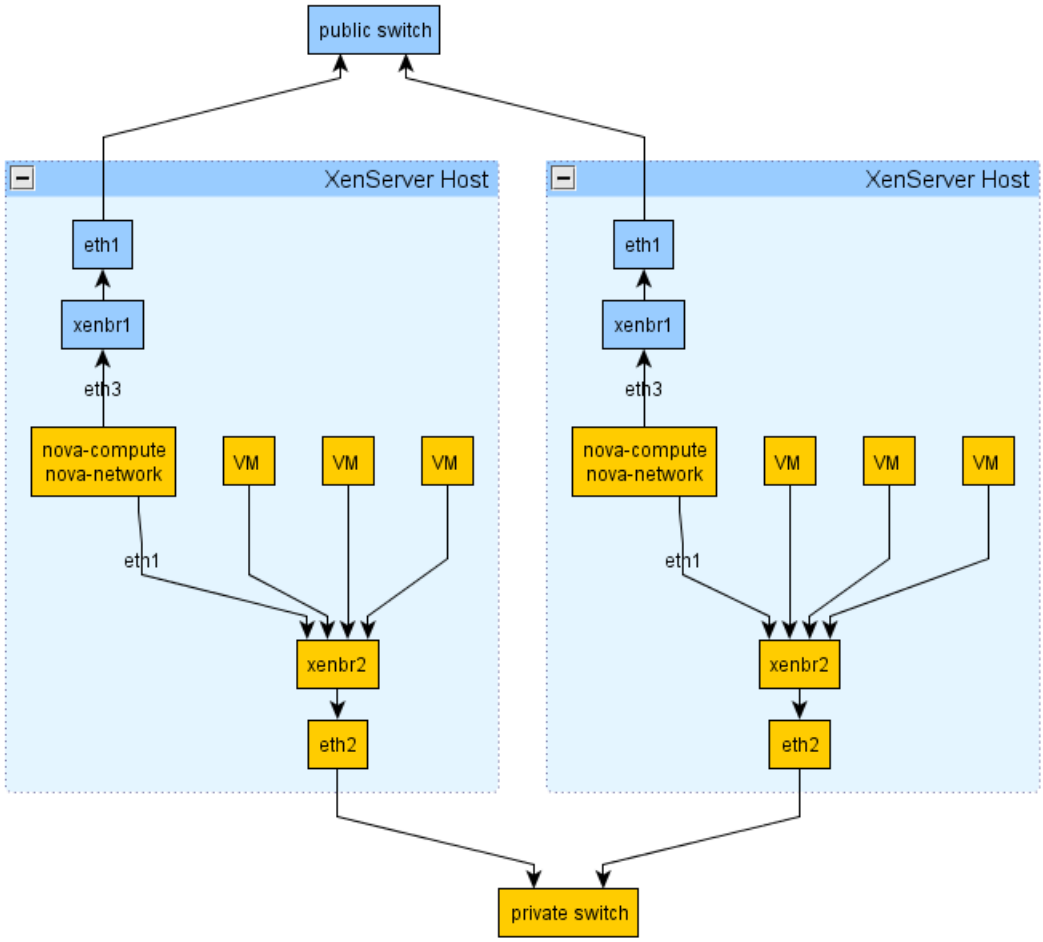
```
dhcpbridge_flagfile=/etc/nova/nova.conf
dhcpbridge=/usr/bin/nova-dhcpbridge
network_manager=nova.network.manager.FlatDHCPManager
fixed_range=10.0.0.0/8
flat_network_bridge=br100
flat_interface=eth2
flat_injected=False
public_interface=eth0
```

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 9.5. 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
flat_interface=eth1
flat_network_bridge=xenbr2
public_interface=eth3
multi_host=True
dhcpbridge_flagfile=/etc/nova/nova.conf
fixed_range=10.0.0.0/24
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 an `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`.

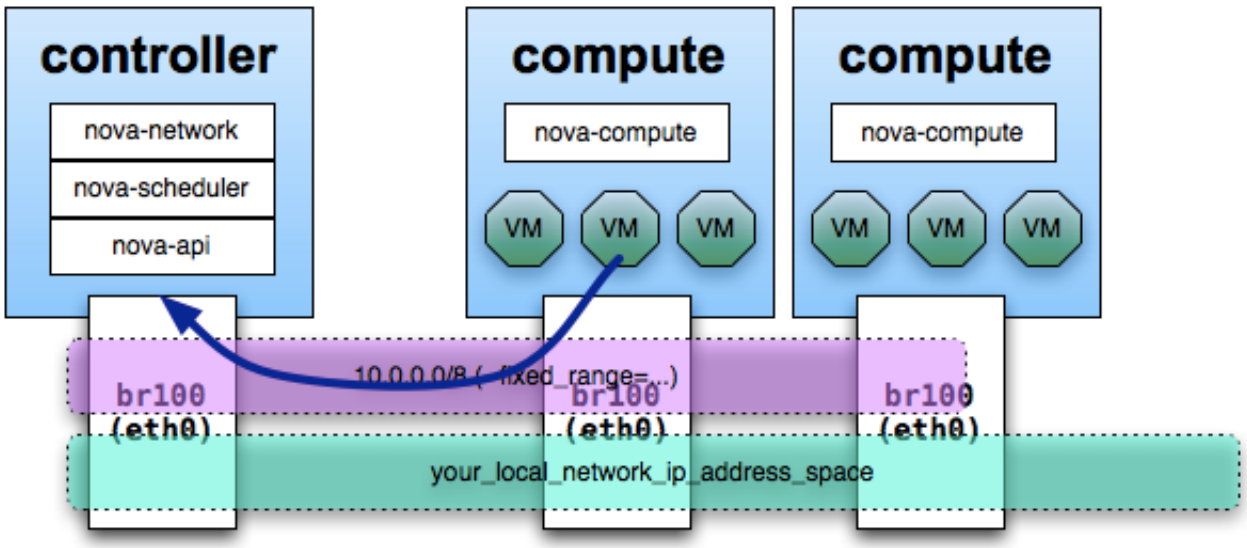
When using XenServer, it is best to use the firewall driver written specifically for XenServer. This pushes the firewall rules down to the hypervisor, rather than running them in the VM that is running `nova-network`.

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 configured 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.

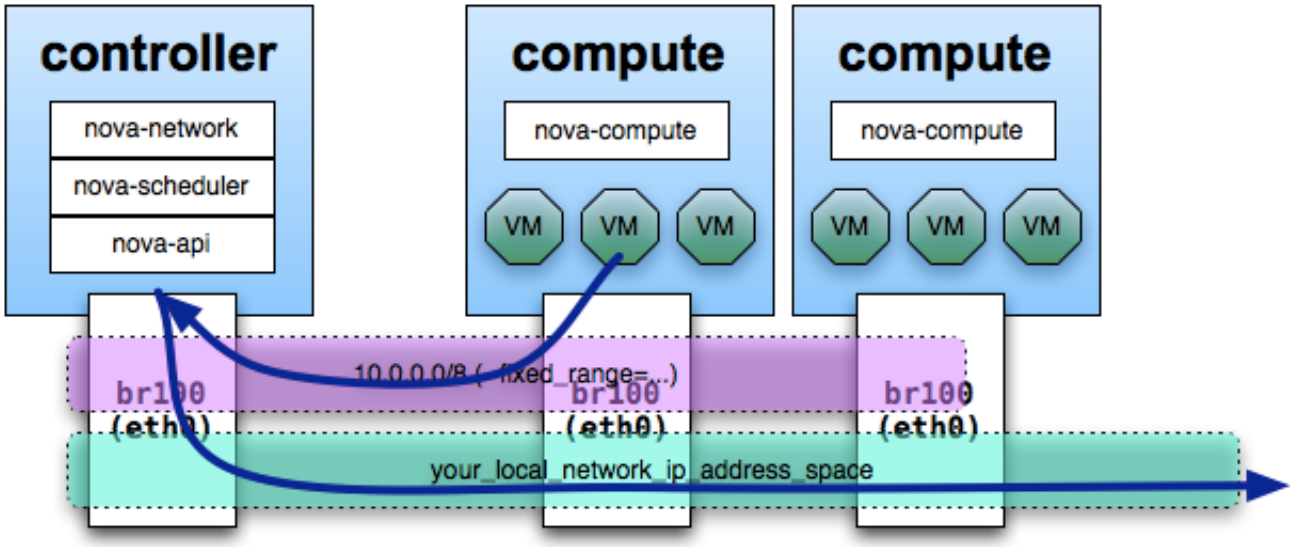
When a virtual machine sends traffic out to the public networks, it sends it first to its default gateway, which is where nova-network is configured.

Figure 9.6. 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 9.7. 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 is 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 overriden at network creation time . The example above uses a network size of 256, whicih corresponds to a `/24` network.

Networks are created with the **nova-manage network create** command. Here is an example of how to create a network consistent with the above example configuration options, as root:

```
# nova-manage network create --label=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.

Instead of manually specifying a VLAN, bridge, and project id, you can create many networks at once and have the Compute service automatically associate these networks with tenants as needed, as well as automatically generating the VLAN IDs and bridge interface names. For example, the following command would create 100 networks, from `172.16.100.0/24` to `172.16.199.0/24`. (This assumes the `network_size=256` option has been set at nova.conf, though this can also be specified by passing `--network_size=256` as a flag to the **nova-manage** command)

```
# nova-manage network create --num_networks=100 --fixed_range_v4=172.16.100.0/
24
```

The **nova-manage network create** command supports many configuration options, which are displayed when called with the `--help` flag:

```
Usage: nova-manage network create <args> [options]

Options:
  -h, --help            show this help message and exit
  --label=<label>       Label for network (ex: public)
  --fixed_range_v4=<x.x.x.x/yy>
                        IPv4 subnet (ex: 10.0.0.0/8)
  --num_networks=<number>
                        Number of networks to create
  --network_size=<number>
                        Number of IPs per network
  --vlan=<vlan id>      vlan id
  --vpn=VPN_START       vpn start
  --fixed_range_v6=FIXED_RANGE_V6
                        IPv6 subnet (ex: fe80::/64)
  --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-mange network create** command can be used to override settings from `nova.conf`:

`--network_size` Overrides the `network_size` configuration option

`--bridge_interface` Overrides the `vlan_interface` configuration option

To view a list of the networks that have been created, as root:

```
# nova-manage network list
```

To modify an existing network, 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
```

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-manage network modify** 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
```

Warning

Due to Compute bug #754900, deleting a network with the **nova-manage network delete** command does not delete the associated fixed IP addresses. As a workaround, these fixed IP addresses can be deleted by connecting to the `nova` database and issuing the SQL query (this example assumes the deleted network id is `1`

```
DELETE from fixed_ips where network_id=1;
```

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
```

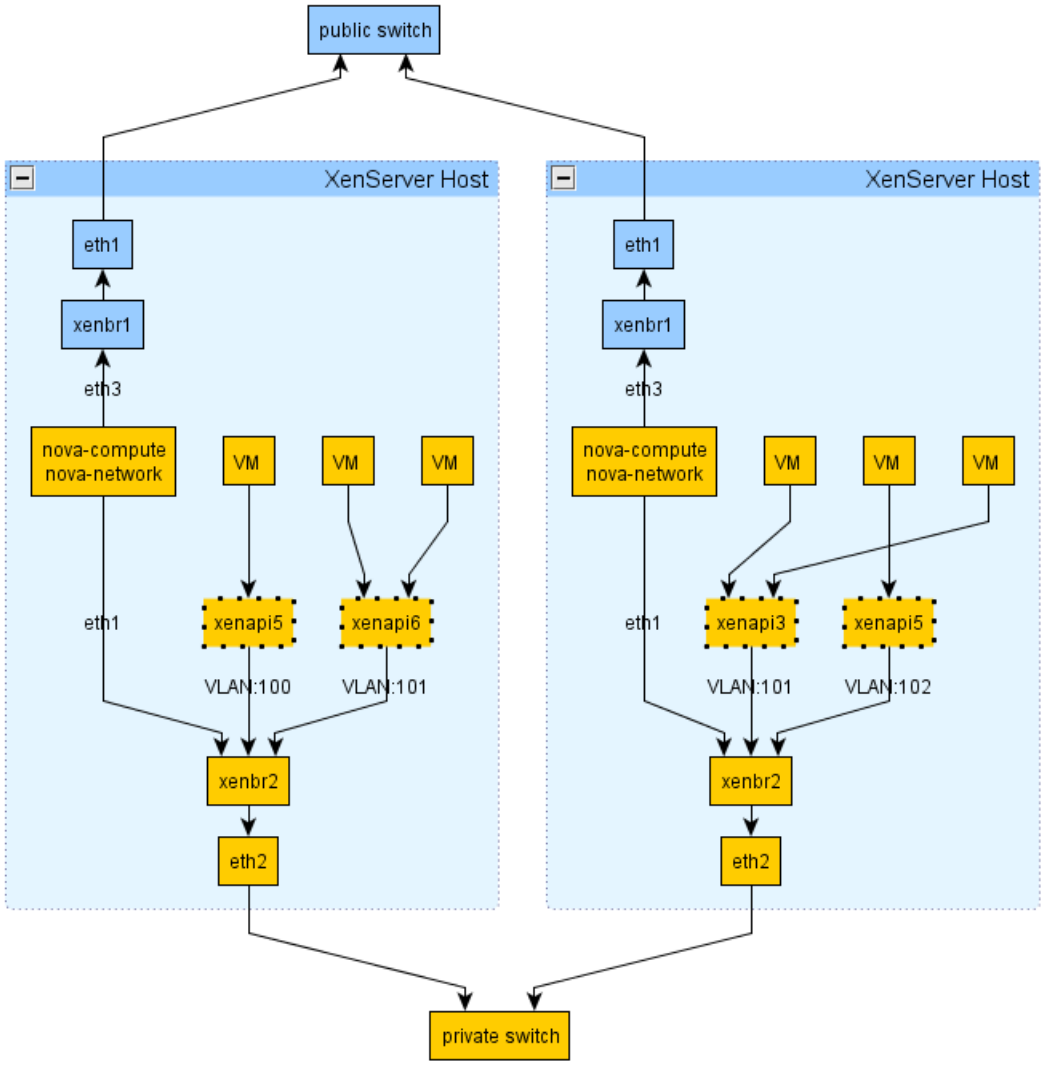
In certain cases, the network manager may not properly tear down bridges and VLANs when it is stopped. If you attempt to restart the network manager and it does not start, check the logs for errors indicating that a bridge device already exists. If this is the case, you will likely need to tear down 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 9.8. 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
network_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 interfce 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 exsists.
- 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 the 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 dnsmaq 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=true
# 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 withing 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 retrive 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 739079a-b-0f8e-404a-ae6e-a91f4fe99c94
```

Make sure the instance has been upload to the Glance repository :

```
$ nova image-list
```

```
+--------------------------------------+---------------+--------
+--------------------------------------+
|                  ID                  |      Name     | Status |
   Server                |
+--------------------------------------+---------------+--------
+--------------------------------------+
| 0bfc8fd3-1590-463b-b178-bce30be5ef7b | cloud-pipance | ACTIVE |
 fb93eda8-4eb8-42f7-b53c-91c6d83cface |
+--------------------------------------+---------------+--------
+--------------------------------------+
```

Make that image public (snapshot-based images are private by default):

```
$ glance 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=0bfc8fd3-1590-463b-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 cloud-pipe 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 cloudpipe 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 cloudpipes 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 cloudpipes 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 cloudpipe has an updated crl, it will block revoked users from connecting to the vpn.

The userdata for cloudpipe isn't currently updated when certs are revoked, so it is necessary to restart the cloudpipe instance if a user's credentials are revoked.

Restarting and Logging into the Cloudpipe VPN

You can reboot a cloudpipe 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 cloudpipe extension. The cloudpipe 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 cloudpipe instance should be in the `keys/<project_id>` folder. You can use this key to log into the cloudpipe 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 cloudpipe image that you create.

Remote access to your cloudpipe instance from an OpenVPN client

Now your cloudpipe instance is running, you can use your favorite OpenVPN client in order to access your instances within their private network cloudpipe is connected to. In these sections we will present both ways of using cloudpipe, the first using a configuration file for clients without interfaces, and for clients using an interface.

Connect to your cloudpipe 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/cloudpipe/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 $cloudpipe-public-ip $cloudpipe-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 cloudpipe instance, you can run the following command :

```
$ nova cloudpipe-list
```

```
+----------------------------------+------------+-------------
+---------------+
|            Project Id            | Public IP  | Public Port |  Internal IP
  |
+----------------------------------+------------+-------------
+---------------+
| d1f5d27ccf594cdbb034c8a4123494e9 | 172.17.1.3 | 1000        | 192.168.22.
34 |
+----------------------------------+------------+-------------
+---------------+
```

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 cloudpipe 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 cloudpipe instance.

   These informations can be found by running a

```
$ nova cloudpipe-list
```

```
+----------------------------------+------------+-------------
+---------------+
|            Project Id            | Public IP  | Public Port |  Internal IP
  |
+----------------------------------+------------+-------------
+---------------+
| d1f5d27ccf594cdbb034c8a4123494e9 | 172.17.1.3 | 1000        | 192.168.22.
34 |
+----------------------------------+------------+-------------
+---------------+
```

Figure 9.9. Configuring Viscosity

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 | id  |
 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 | 484 |
 192.168.22.34 |         13 |         630 |         0 |      0 |        1 |
               NULL | NULL |
+---------------------+---------------------+------------+---------+-----
+---------------+------------+-------------+-----------+--------+----------
+----------------------+------+
```

```
(mysql) UPDATE fixed_ips SET allocated=0, leased=0, instance_id=NULL WHERE
 address='192.168.22.34';
```

```
(mysql) SELECT * FROM fixed_ips WHERE address='192.168.22.34';
```

```
 +---------------------+---------------------+------------+---------+-----
+--------------+-----------+------------+-----------+--------+----------
+----------------------+------+
| 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 | 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 -s 0.0.0.0/0
$ nova secgroup-add-rule default tcp 22 22 -s 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.

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, 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 --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 [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. You can disassociate the project from the network with a scrub command and the project name as the final parameter:

```
$ nova-manage project scrub projectname
```

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 9.10. multinic flat manager

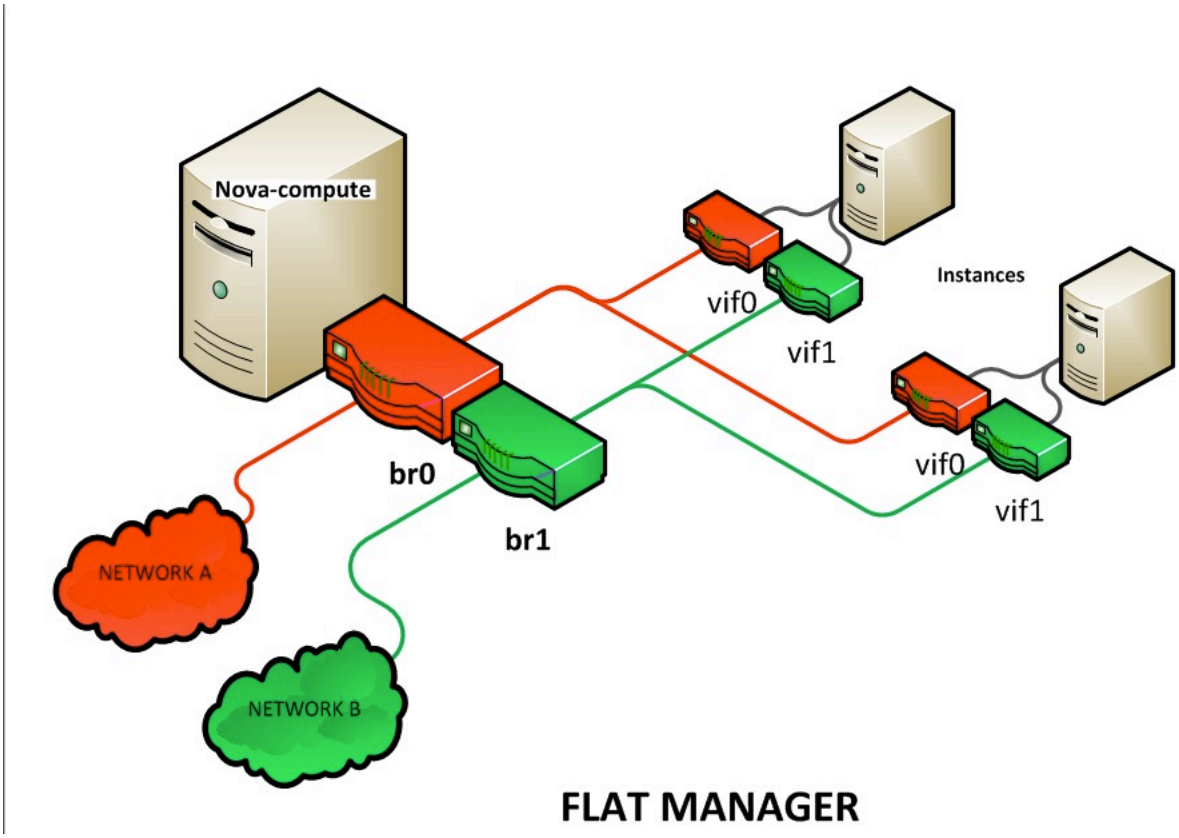


Figure 9.11. multinic flatdhcp manager

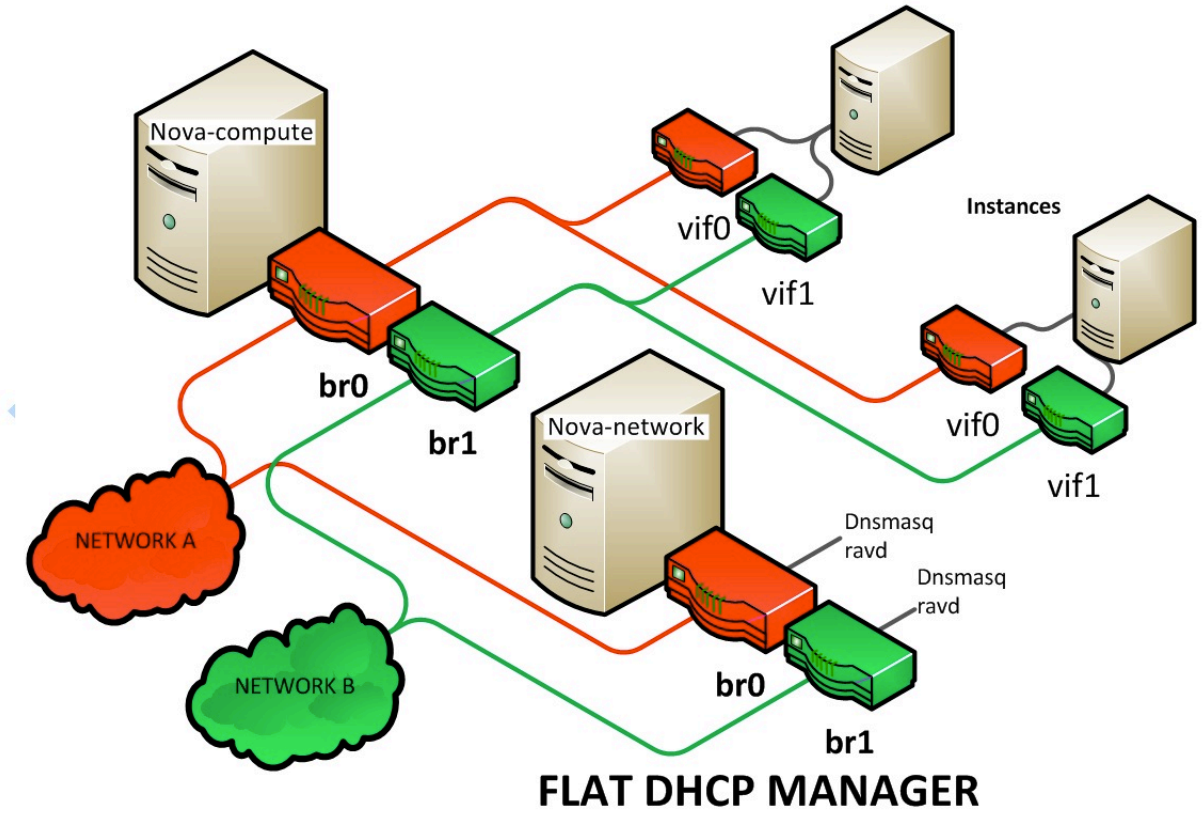
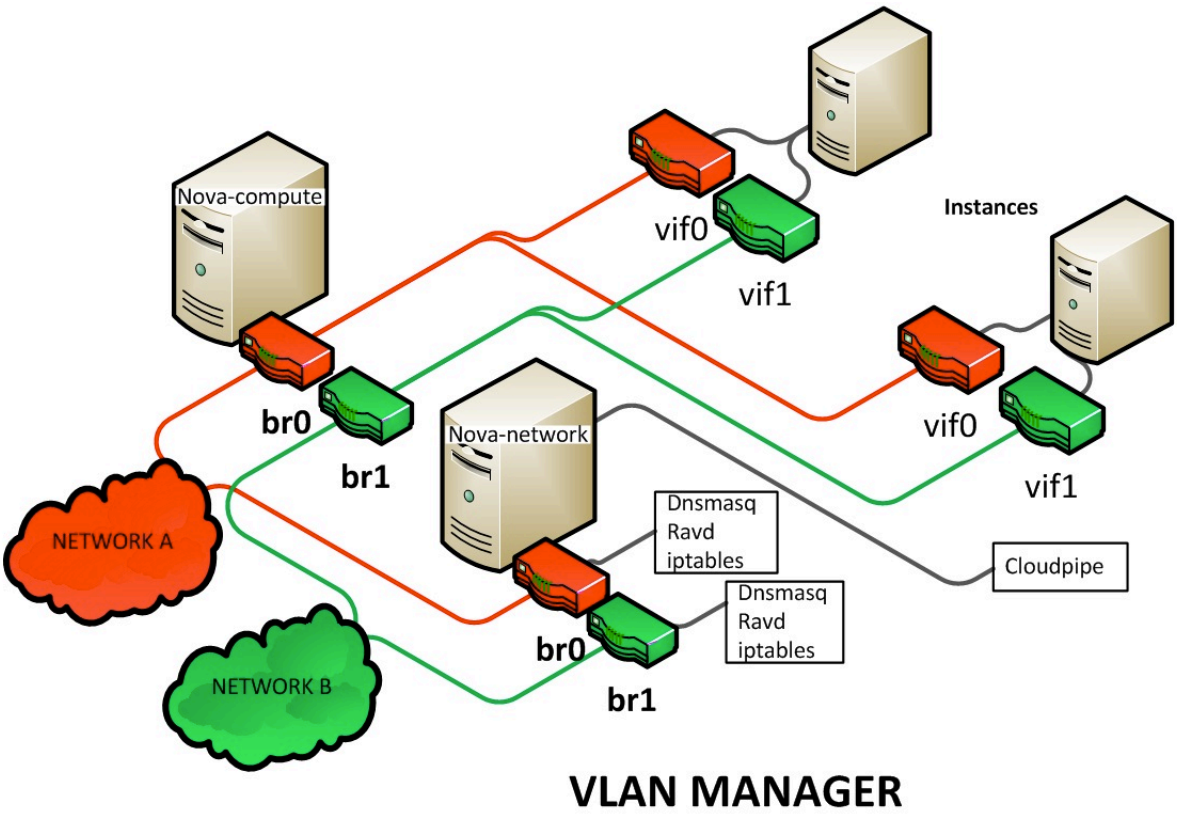


Figure 9.12. multinic VLAN manager



Using the multinic feature

The first thing to do is to create a new network and attach it to your project :

```
$ nova-manage network create --fixed_range_v4=20.20.0.0/24 --num_networks=1 --
network_size=256 --label=test --project=$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=20.10.0.14 |
+-----+------------+--------+---------------------------------------+
```

Note

Make sure to power up the second interface on the instance, otherwise that last won't be reacheable via its second IP. Here is an example of how to setup the interfaces within the instance :

`/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
```

Existing High Availability Options for Networking

Adapted from a blog post byVish 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.

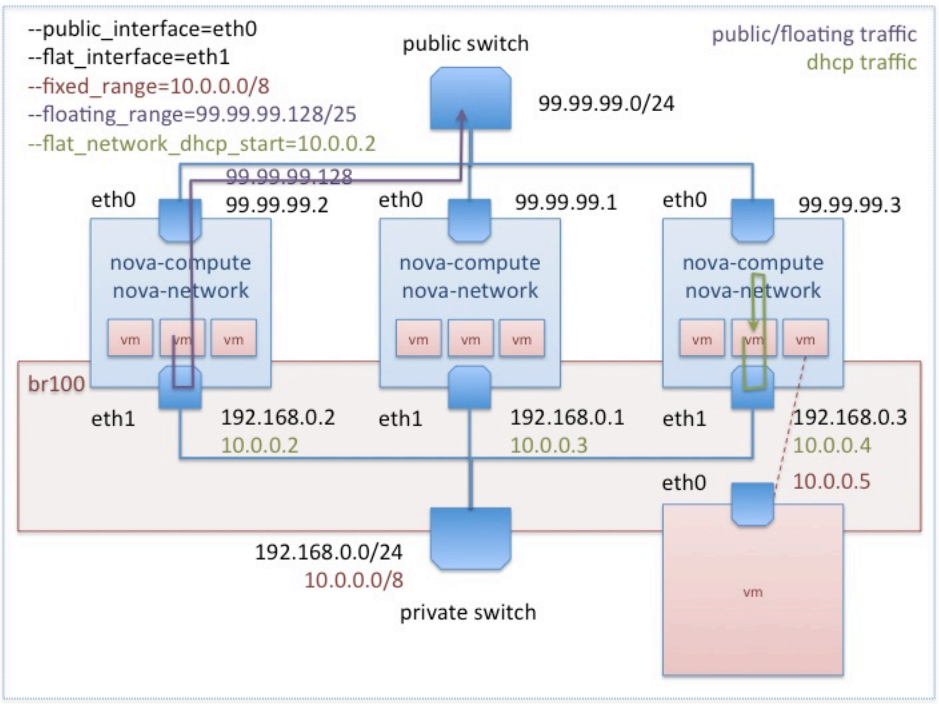
HA Option 1: Multi-host

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 9.13. 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
```

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

The `multi_host` option must be in place for network creation 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 VM is on. You need to set the configuration option `enabled_apis` such that it includes `metadata` in the list of enabled APIs.

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 using the following syntax:

   ```
   dhcpoption=3,<ip of gateway>
   ```

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.

Warning

The hardware gateway option is not available when running in VLAN mode, as it can only be used to specify a single gateway for all instances, and VLAN mode requires a separate gateway for each network.

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 -nv -t nat

-A nova-network-PREROUTING -d 68.99.26.170/32 -j DNAT --to-destination10.0.0.3
-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 revesrse 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.

10. Volumes

Managing Volumes

Nova-volume is the service that allows you to give extra block level storage to your OpenStack Compute instances. You may recognize this as a similar offering from Amazon EC2 known as Elastic Block Storage (EBS). However, nova-volume is not the same implementation that EC2 uses today. Nova-volume is an iSCSI solution that employs the use of Logical Volume Manager (LVM) for Linux. Note that a volume may only be attached to one instance at a time. This is not a 'shared storage' solution like a SAN of NFS on which multiple servers can attach to.

Before going any further; let's discuss the nova-volume implementation in OpenStack:

The nova-volumes service uses iSCSI-exposed LVM volumes to the compute nodes which run instances. Thus, there are two components involved:

1. lvm2, which works with a VG called "nova-volumes" (Refer to http://en.wikipedia.org/wiki/Logical_Volume_Manager_(Linux) for further details)

2. open-iscsi, the iSCSI implementation which manages iSCSI sessions on the compute nodes

Here is what happens from the volume creation to its attachment:

1. The volume is created via **nova volume-create**; which creates an LV into the volume group (VG) "nova-volumes"

2. The volume is attached to an instance via **nova volume-attach**; which creates a unique iSCSI IQN that will be exposed to the compute node

3. The compute node which run the concerned instance has now an active ISCSI session; and a new local storage (usually a /dev/sdX disk)

4. libvirt uses that local storage as a storage for the instance; the instance get a new disk (usually a /dev/vdX disk)

For this particular walkthrough, there is one cloud controller running nova-api, nova-scheduler, nova-objectstore, nova-network and nova-volume services. There are two additional compute nodes running nova-compute. The walkthrough uses a custom partitioning scheme that carves out 60GB of space and labels it as LVM. The network is a /28 .80-.95, and FlatManger is the NetworkManager setting for OpenStack Compute (Nova).

Please note that the network mode doesn't interfere at all with the way nova-volume works, but networking must be set up for for nova-volumes to work. Please refer to Networking for more details.

To set up Compute to use volumes, ensure that nova-volume is installed along with lvm2. The guide will be split in four parts :

- A- Installing the nova-volume service on the cloud controller.

- B- Configuring the "nova-volumes" volume group on the compute nodes.
- C- Troubleshooting your nova-volume installation.
- D- Backup your nova volumes.

A- Install nova-volume on the cloud controller.

This is simply done by installing the two components on the cloud controller :

```
$ apt-get install lvm2 nova-volume
```

- **Configure Volumes for use with nova-volume**

  If you do not already have LVM volumes on hand, but have free drive space, you will need to create a LVM volume before proceeding. Here is a short run down of how you would create a LVM from free drive space on your system. Start off by issuing an fdisk command to your drive with the free space:

  ```
  $ fdisk /dev/sda
  ```

  Once in fdisk, perform the following commands:

  1. Press **n** to create a new disk partition,
  2. Press **p** to create a primary disk partition,
  3. Press **1** to denote it as 1st disk partition,
  4. Either press ENTER twice to accept the default of 1st and last cylinder – to convert the remainder of hard disk to a single disk partition -OR- press ENTER once to accept the default of the 1st, and then choose how big you want the partition to be by specifying `+size[K,M,G]` e.g. +5G or +6700M.
  5. Press **t** and select the new partition that you have created.
  6. Press **8e** change your new partition to 8e, i.e. Linux LVM partition type.
  7. Press **p** to display the hard disk partition setup. Please take note that the first partition is denoted as `/dev/sda1` in Linux.
  8. Press **w** to write the partition table and exit fdisk upon completion.

     Refresh your partition table to ensure your new partition shows up, and verify with **fdisk**. We then inform the OS about the table partition update :

     ```
     $ partprobe
     $ fdisk -l
     ```

     You should see your new partition in this listing.

Here is how you can set up partitioning during the OS install to prepare for this nova-volume configuration:

```
root@osdemo03:~# fdisk -l
```

```
Device Boot Start End Blocks Id System

/dev/sda1 * 1 12158 97280 83 Linux
/dev/sda2 12158 24316 97655808 83 Linux

/dev/sda3 24316 24328 97654784 83 Linux
/dev/sda4 24328 42443 145507329 5 Extended

/dev/sda5 24328 32352 64452608 8e Linux LVM
/dev/sda6 32352 40497 65428480 8e Linux LVM

/dev/sda7 40498 42443 15624192 82 Linux swap / Solaris
```

Now that you have identified a partition has been labeled for LVM use, perform the following steps to configure LVM and prepare it as nova-volumes. **You must name your volume group 'nova-volumes' or things will not work as expected**:

```
$ pvcreate /dev/sda5
$ vgcreate nova-volumes /dev/sda5
```

B- Configuring nova-volume on the compute nodes

Since you have created the volume group, you will be able to use the following tools for managing your volumes:

nova volume-create

nova volume-attach

nova volume-detach

nova volume-delete

Note

If you are using KVM as your hypervisor, then the actual device name in the guest will be different than the one specified in the nova volume-attach command. You can specify a device name to the KVM hypervisor, but the actual means of attaching to the guest is over a virtual PCI bus. When the guest sees a new device on the PCI bus, it picks the next available name (which in most cases is /dev/vdc) and the disk shows up there on the guest.

- **Installing and configuring the iSCSI initiator**

  Remember that every node will act as the iSCSI initiator while the server running nova-volumes will act as the iSCSI target. So make sure, before going further that your nodes

can communicate with you nova-volumes server. If you have a firewall running on it, make sure that the port 3260 (tcp) accepts incoming connections.

First install the open-iscsi package on the initiators, so on the compute-nodes **only**

```
$ apt-get install open-iscsi
```

Then on the target, which is in our case the cloud-controller, the iscsitarget package :

```
$ apt-get install iscsitarget
```

This package could refuse to start with a "FATAL: Module iscsi_trgt not found" error.

This error is caused by the kernel which does not contain the iscsi module's source into it ; you can install the kernel modules by installing an extra package :

```
$ apt-get install iscsitarget-dkms
```

(the Dynamic Kernel Module Support is a framework used for created modules with non-existent sources into the current kernel)

You have to enable it so the startut script (/etc/init.d/iscsitarget) can start the daemon:

```
$ sed -i 's/false/true/g' /etc/default/iscsitarget
```

Then run on the nova-controller (iscsi target) :

```
$ service iscsitarget start
```

And on the compute-nodes (iscsi initiators) :

```
$ service open-iscsi start
```

- **Start nova-volume and create volumes**

  You are now ready to fire up nova-volume, and start creating volumes!

```
$ service nova-volume start
```

  Once the service is started, login to your controller and ensure you've properly sourced your 'novarc' file.

  One of the first things you should do is make sure that nova-volume is checking in as expected. You can do so using nova-manage:

```
$ nova-manage service list
```

If you see a smiling 'nova-volume' in there, you are looking good. Now create a new volume:

```
$ nova volume-create --display_name myvolume 10
```

–display_name sets a readable name for the volume, while the final argument refers to the size of the volume in GB.

You should get some output similar to this:

```
                        +----+-----------+--------------+------
+-------------+-----------------------------------+
                        | ID |   Status  | Display Name | Size | Volume Type
 |             Attached to              |
                        +----+-----------+--------------+------
+-------------+-----------------------------------+
                        | 1  | available | myvolume     | 10   | None
 |                                  |
                        +----+-----------+--------------+------
+-------------+-----------------------------------+
```

You can view that status of the volumes creation using **nova volume-list**. Once that status is 'available,' it is ready to be attached to an instance:

```
$ nova volume-attach 857d70e4-35d5-4bf6-97ed-bf4e9a4dcf5a 1 /dev/vdb
```

The first argument refers to the instance you will attach the volume to; The second is the volume ID; The third is the mountpoint **on the compute-node** that the volume will be attached to

By doing that, the compute-node which runs the instance basically performs an iSCSI connection and creates a session. You can ensure that the session has been created by running :

```
$ iscsiadm -m session
```

Which should output :

```
root@nova-cn1:~# iscsiadm -m session
tcp: [1] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-1
```

If you do not get any errors, you can login to the instance and see if the new space is there.

KVM changes the device name, since it's not considered to be the same type of device as the instances uses as it's local one, you will find the nova-volume will be designated as "/dev/vdX" devices, while local are named "/dev/sdX".

You can check the volume attachment by running :

```
$ dmesg | tail
```

You should from there see a new disk. Here is the output from **fdisk -l**:

```
Disk /dev/vda: 10.7 GB, 10737418240 bytes
16 heads, 63 sectors/track, 20805 cylinders
Units = cylinders of 1008 * 512 = 516096 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes
Disk identifier: 0×00000000
Disk /dev/vda doesn't contain a valid partition table
Disk /dev/vdb: 21.5 GB, 21474836480 bytes <––Here is our new volume!
16 heads, 63 sectors/track, 41610 cylinders
Units = cylinders of 1008 * 512 = 516096 bytes
Sector size (logical/physical): 512 bytes / 512 bytes
I/O size (minimum/optimal): 512 bytes / 512 bytes Disk identifier:
 0×00000000
```

Now with the space presented, let's configure it for use:

```
$ fdisk /dev/vdb
```

1. Press **n** to create a new disk partition.
2. Press **p** to create a primary disk partition.
3. Press **1** to designated it as the first disk partition.
4. Press ENTER twice to accept the default of first and last cylinder – to convert the remainder of hard disk to a single disk partition.
5. Press **t**, then select the new partition you made.
6. Press **83** change your new partition to 83, i.e. Linux partition type.
7. Press **p** to display the hard disk partition setup. Please take note that the first partition is denoted as /dev/vda1 in your instance.
8. Press **w** to write the partition table and exit fdisk upon completion.
9. Lastly, make a file system on the partition and mount it.

```
$ mkfs.ext3 /dev/vdb1
$ mkdir /extraspace
```

```
$ mount /dev/vdb1 /extraspace
```

Your new volume has now been successfully mounted, and is ready for use! The commands are pretty self-explanatory, so play around with them and create new volumes, tear them down, attach and reattach, and so on.

C- Troubleshoot your nova-volume installation

If the volume attachment doesn't work, you should be able to perform different checks in order to see where the issue is. The nova-volume.log and nova-compute.log will help you to diagnosis the errors you could encounter :

nova-compute.log / nova-volume.log

- *ERROR "Cannot resolve host"*

```
(nova.root): TRACE: ProcessExecutionError: Unexpected error while running
 command.
(nova.root): TRACE: Command: sudo iscsiadm -m discovery -t sendtargets -p
 ubuntu03c
(nova.root): TRACE: Exit code: 255
(nova.root): TRACE: Stdout: ''
(nova.root): TRACE: Stderr: 'iscsiadm: Cannot resolve host ubuntu03c.
 getaddrinfo error: [Name or service not known]\n\niscsiadm:
cannot resolve host name ubuntu03c\niscsiadm: Could not perform SendTargets
 discovery.\n'
(nova.root): TRACE:
```

This error happens when the compute node is unable to resolve the nova-volume server name. You could either add a record for the server if you have a DNS server; or add it into the `/etc/hosts` file of the nova-compute.

- *ERROR "No route to host"*

```
iscsiadm: cannot make connection to 172.29.200.37: No route to host\
niscsiadm: cannot make connection to 172.29.200.37
```

This error could be caused by several things, but **it means only one thing : openiscsi is unable to establish a communication with your nova-volumes server**.

The first thing you could do is running a telnet session in order to see if you are able to reach the nova-volume server. From the compute-node, run :

```
$ telnet $ip_of_nova_volumes  3260
```

If the session times out, check the server firewall ; or try to ping it. You could also run a tcpdump session which may also provide extra information :

```
$ tcpdump -nvv -i $iscsi_interface port dest $ip_of_nova_volumes
```

```
```

Again, try to manually run an iSCSI discovery via :

```
$ iscsiadm -m discovery -t st -p $ip_of_nova-volumes
```

- *"Lost connectivity between nova-volumes and node-compute ; how to restore a clean state ?"*

  Network disconnection can happens, from an "iSCSI view", losing connectivity could be seen as a physical removal of a server's disk. If the instance runs a volume while you loose the network between them, you won't be able to detach the volume. You would encounter several errors. Here is how you could clean this :

  First, from the nova-compute, close the active (but stalled) iSCSI session, refer to the volume attached to get the session, and perform the following command :

```
$ iscsiadm -m session -r $session_id -u
```

  Here is an **iscsi -m** session output :

```
tcp: [1] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-1
tcp: [2] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-2
tcp: [3] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-3
tcp: [4] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-4
tcp: [5] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-5
tcp: [6] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-6
tcp: [7] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-7
tcp: [9] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-9
```

  For example, to free volume 9, close the session number 9.

  The cloud-controller is actually unaware of the iSCSI session closing, and will keeps the volume state as `in-use`:

```
                            +----+-----------+--------------+------
+-------------+--------------------------------------+
                            | ID |   Status  | Display Name | Size | Volume
 Type |             Attached to              |
                            +----+-----------+--------------+------
+-------------+--------------------------------------+
                            | 9  | in-use    | New Volume   | 20   | None
     | 7db4cb64-7f8f-42e3-9f58-e59c9a31827d |
```

  You now have to inform the cloud-controller that the disk can be used. Nova stores the volumes info into the "volumes" table. You will have to update four fields into the database nova uses (eg. MySQL). First, conect to the database :

```
$ mysql -uroot -p$password nova
```

Using the volume id, you will have to run the following sql queries

```
mysql> update volumes set mountpoint=NULL where id=9;
mysql> update volumes set status="available" where status "error_deleting"
 where id=9;
mysql> update volumes set attach_status="detached" where id=9;
mysql> update volumes set instance_id=0 where id=9;
```

Now if you run again **nova volume-list** from the cloud controller, you should see an available volume now :

```
                           +----+-----------+--------------+------
+-------------+--------------------------------------+
                           | ID |   Status  | Display Name | Size | Volume
 Type |             Attached to              |
                           +----+-----------+--------------+------
+-------------+--------------------------------------+
                           | 9  | available  | New Volume   | 20   | None
     |                                  |
```

You can now proceed to the volume attachment again!

D- Backup your nova-volume disks

While Diablo provides the snapshot functionality (using LVM snapshot), you can also back up your volumes. The advantage of this method is that it reduces the size of the backup; only existing data will be backed up, instead of the entire volume. For this example, assume that a 100 GB nova-volume has been created for an instance, while only 4 gigabytes are used. This process will back up only those 4 giga-bytes, with the following tools:

1. **lvm2**, directly manipulates the volumes.
2. **kpartx** discovers the partition table created inside the instance.
3. **tar** creates a minimum-sized backup
4. **sha1sum** calculates the backup checksum, to check its consistency

1- Create a snapshot of a used volume

- In order to backup our volume, we first need to create a snapshot of it. An LVM snapshot is the exact copy of a logical volume, which contains data in a frozen state. This prevents data corruption, because data will not be manipulated during the process of creating the volume itself. Remember the volumes created through a **nova volume-create** exist in an LVM's logical volume.

  Before creating the snapshot, ensure that you have enough space to save it. As a precaution, you should have at least twice as much space as the potential snapshot size. If insufficient space is available, there is a risk that the snapshot could become corrupted.

  Use the following command to obtain a list of all volumes.

```
$ lvdisplay
```

In this example, we will refer to a volume called `volume-00000001`, which is a 10GB volume. This process can be applied to all volumes, not matter their size. At the end of the section, we will present a script that you could use to create scheduled backups. The script itself exploits what we discuss here.

First, create the snapshot; this can be achieved while the volume is attached to an instance :

```
$ lvcreate --size 10G --snapshot --name volume-00000001-snapshot /dev/nova-
volumes/volume-00000001
```

We indicate to LVM we want a snapshot of an already existing volume with the `--snapshot` configuration option. The command includes the size of the space reserved for the snapshot volume, the name of the snapshot, and the path of an already existing volume (In most cases, the path will be `/dev/nova-volumes/$volume_name`).

The size doesn't have to be the same as the volume of the snapshot. The size parameter designates the space that LVM will reserve for the snapshot volume. As a precaution, the size should be the same as that of the original volume, even if we know the whole space is not currently used by the snapshot.

We now have a full snapshot, and it only took few seconds !

Run **lvdisplay** again to verify the snapshot. You should see now your snapshot :

```
                --- Logical volume ---
  LV Name                /dev/nova-volumes/volume-00000001
  VG Name                nova-volumes
  LV UUID                gI8hta-p21U-IW2q-hRN1-nTzN-UC2G-dKbdKr
  LV Write Access        read/write
  LV snapshot status     source of
                         /dev/nova-volumes/volume-00000026-snap [active]
  LV Status              available
  # open                 1
  LV Size                15,00 GiB
  Current LE             3840
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           251:13

  --- Logical volume ---
  LV Name                /dev/nova-volumes/volume-00000001-snap
  VG Name                nova-volumes
  LV UUID                HlW3Ep-g5I8-KGQb-IRvi-IRYU-lIKe-wE9zYr
  LV Write Access        read/write
  LV snapshot status     active destination for /dev/nova-volumes/
volume-00000026
  LV Status              available
```

```
  # open                 0
  LV Size                15,00 GiB
  Current LE             3840
  COW-table size         10,00 GiB
  COW-table LE           2560
  Allocated to snapshot  0,00%
  Snapshot chunk size    4,00 KiB
  Segments               1
  Allocation             inherit
  Read ahead sectors     auto
  - currently set to     256
  Block device           251:14
```

2- Partition table discovery

- If we want to exploit that snapshot with the **tar** program, we first need to mount our partition on the nova-volumes server.

  kpartx is a small utility which performs table partition discoveries, and maps it. It can be used to view partitions created inside the instance. Without using the partitions created inside instances, we won' t be able to see its content and create efficient backups.

```
$ kpartx -av /dev/nova-volumes/volume-00000001-snapshot
```

  If no errors are displayed, it means the tools has been able to find it, and map the partition table. Note that on a Debian flavor distro, you could also use **apt-get install kpartx**.

  You can easily check the partition table map by running the following command:

```
$ ls /dev/mapper/nova*
```

  You should now see a partition called `nova--volumes-volume--00000001--snapshot1`

  If you created more than one partition on that volumes, you should have accordingly several partitions; for example. `nova--volumes-volume--00000001--snapshot2`, `nova--volumes-volume--00000001--snapshot3` and so forth.

  We can now mount our partition :

```
$ mount /dev/mapper/nova--volumes-volume--volume--00000001--snapshot1 /mnt
```

  If there are no errors, you have successfully mounted the partition.

  You should now be able to directly access the data that were created inside the instance. If you receive a message asking you to specify a partition, or if you are unable to mount it (despite a well-specified filesystem) there could be two causes :

  - You didn't allocate enough space for the snapshot

- **kpartx** was unable to discover the partition table.

Allocate more space to the snapshot and try the process again.

3- Use tar in order to create archives

- Now that the volume has been mounted, you can create a backup of it :

```
$ tar --exclude={"lost+found","some/data/to/exclude"} -czf volume-00000001.
tar.gz -C /mnt/ /backup/destination
```

This command will create a tar.gz file containing the data, *and data only*. This ensures that you do not waste space by backing up empty sectors.

4- Checksum calculation I

- You should always have the checksum for your backup files. The checksum is a unique identifier for a file.

  When you transfer that same file over the network, you can run another checksum calculation. If the checksums are different, this indicates that the file is corrupted; thus, the checksum provides a method to ensure your file has not been corrupted during its transfer.

  The following command runs a checksum for our file, and saves the result to a file :

```
$ sha1sum volume-00000001.tar.gz > volume-00000001.checksum
```

  Be aware the **sha1sum** should be used carefully, since the required time for the calculation is directly proportional to the file's size.

  For files larger than ~4-6 gigabytes, and depending on your CPU, the process may take a long time.

5- After work cleaning

- Now that we have an efficient and consistent backup, the following commands will clean up the file system.

  1. Unmount the volume: **unmount /mnt**
  2. Delete the partition table: **kpartx -dv /dev/nova-volumes/volume-00000001-snapshot**
  3. Remove the snapshot: **lvremove -f /dev/nova-volumes/volume-00000001-snapshot**

  And voila :) You can now repeat these steps for every volume you have.

6- Automate your backups

Because you can expect that more and more volumes will be allocated to your nova-volume service, you may want to automate your backups. This script here will assist you on this

task. The script performs the operations from the previous example, but also provides a mail report and runs the backup based on the `backups_retention_days` setting. It is meant to be launched from the server which runs the nova-volumes component.

Here is an example of a mail report:

```
Backup Start Time - 07/10 at 01:00:01
Current retention - 7 days

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-00000019/
volume-00000019_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-00000019 - 0 h 1 m and 21 seconds. Size - 3,5G

The backup volume is mounted. Proceed...
Removing old backups...  : /BACKUPS/EBS-VOL/volume-0000001a/
volume-0000001a_28_09_2011.tar.gz
     /BACKUPS/EBS-VOL/volume-0000001a - 0 h 4 m and 15 seconds. Size - 6,9G
---------------------------------------
Total backups size - 267G - Used space : 35%
Total execution time - 1 h 75 m and 35 seconds
```

The script also provides the ability to SSH to your instances and run a mysqldump into them. In order to make this to work, ensure the connection via the nova's project keys is enabled. If you don't want to run the mysqldumps, you can turn off this functionality by adding `enable_mysql_dump=0` to the script.

Volume drivers

The default nova-volume behaviour can be altered by using different volume drivers that are included in Nova codebase. To set volume driver, use `volume_driver` flag. The default is as follows:

```
--volume_driver=nova.volume.driver.ISCSIDriver
```

Ceph RADOS block device (RBD)

If you are using KVM or QEMU as your hypervisor, the Compute service can be configured to use Ceph's RADOS block devices (RBD) for volumes. Add the following lines to nova.conf on the host the runs the **nova-volume** service to enable the RBD driver:

```
volume_driver=nova.volume.driver.RBDDriver
rbd_pool=nova
```

Nexenta

NexentaStor Appliance is NAS/SAN software platform designed for building reliable and fast network storage arrays. The NexentaStor is based on the OpenSolaris and uses ZFS as a disk management system. NexentaStor can serve as a storage node for the OpenStack and provide block-level volumes for the virtual servers via iSCSI protocol.

The Nexenta driver allows you to use Nexenta SA to store Nova volumes. Every Nova volume is represented by a single zvol in a predefined Nexenta volume. For every new volume the driver creates a iSCSI target and iSCSI target group that are used to access it from compute hosts.

To use Nova with Nexenta Storage Appliance, you should:

- set `--volume_driver=nova.volume.nexenta.volume.NexentaDriver`.
- set `--nexenta_host` flag to the hostname or IP of your NexentaStor
- set `--nexenta_user` and `--nexenta_password` to the username and password of the user with all necessary privileges on the appliance, including the access to REST API
- set `--nexenta_volume` to the name of the volume on the appliance that you would like to use in Nova, or create a volume named `nova` (it will be used by default)

Nexenta driver has a lot of tunable flags. Some of them you might want to change:

- `nexenta_target_prefix` defines the prefix that will be prepended to volume id to form target name on Nexenta
- `nexenta_target_group_prefix` defines the prefix for target groups
- `nexenta_blocksize` can be set to the size of the blocks in newly created zvols on appliance, with the suffix; for example, the default 8K means 8 kilobytes
- `nexenta_sparse` is boolean and can be set to use sparse zvols to save space on appliance

Some flags that you might want to keep with the default values:

- `nexenta_rest_port` is the port where Nexenta listens for REST requests (the same port where the NMV works)
- `nexenta_rest_protocol` can be set to `http` or `https`, but the default is `auto` which makes the driver try to use HTTP and switch to HTTPS in case of failure
- `nexenta_iscsi_target_portal_port` is the port to connect to Nexenta over iSCSI

Using the XenAPI Storage Manager Volume Driver

The Xen Storage Manager Volume driver (xensm) is a XenAPI hypervisor specific volume driver, and can be used to provide basic storage functionality, including volume creation and destruction, on a number of different storage back-ends. It also enables the capability of using more sophisticated storage back-ends for operations like cloning/snapshots, etc. The list below shows some of the storage plugins already supported in Citrix XenServer and Xen Cloud Platform (XCP):

1. NFS VHD: Storage repository (SR) plugin which stores disks as Virtual Hard Disk (VHD) files on a remote Network File System (NFS).
2. Local VHD on LVM: SR plugin which represents disks as VHD disks on Logical Volumes (LVM) within a locally-attached Volume Group.

3. HBA LUN-per-VDI driver: SR plugin which represents Logical Units (LUs) as Virtual Disk Images (VDIs) sourced by host bus adapters (HBAs). E.g. hardware-based iSCSI or FC support.

4. NetApp: SR driver for mapping of LUNs to VDIs on a NETAPP server, providing use of fast snapshot and clone features on the filer.

5. LVHD over FC: SR plugin which represents disks as VHDs on Logical Volumes within a Volume Group created on an HBA LUN. E.g. hardware-based iSCSI or FC support.

6. iSCSI: Base ISCSI SR driver, provides a LUN-per-VDI. Does not support creation of VDIs but accesses existing LUNs on a target.

7. LVHD over iSCSI: SR plugin which represents disks as Logical Volumes within a Volume Group created on an iSCSI LUN.

8. EqualLogic: SR driver for mapping of LUNs to VDIs on a EQUALLOGIC array group, providing use of fast snapshot and clone features on the array.

Design and Operation

Definitions

- **Backend:** A term for a particular storage backend. This could be iSCSI, NFS, Netapp etc.
- **Backend-config:** All the parameters required to connect to a specific backend. For e.g. For NFS, this would be the server, path, etc.
- **Flavor:** This term is equivalent to volume "types". A user0friendly term to specify some notion of quality of service. For example, "gold" might mean that the volumes will use a backend where backups are possible. A flavor can be associated with multiple backends. The volume scheduler, with the help of the driver, will decide which backend will be used to create a volume of a particular flavor. Currently, the driver uses a simple "first-fit" policy, where the first backend that can successfully create this volume is the one that is used.

Operation

The admin uses the the nova-manage command detailed below to add flavors and backends.

One or more nova-volume service instances will be deployed per availability zone. When an instance is started, it will create storage repositories (SRs) to connect to the backends available within that zone. All nova-volume instances within a zone can see all the available backends. These instances are completely symmetric and hence should be able to service any `create_volume` request within the zone.

Configuring XenAPI Storage Manager

Prerequisites

1. xensm requires that you use either Citrix XenServer or XCP as the hypervisor. The Netapp and EqualLogic backends are not supported on XCP.

2. Ensure all **hosts** running volume and compute services have connectivity to the storage system.

Configuration

- **Set the following configuration options for the nova volume service: (nova-compute also requires the volume_driver configuration option.)**

```
--volume_driver="nova.volume.xensm.XenSMDriver"
--use_local_volumes=False
```

- **The backend configs that the volume driver uses need to be created before starting the volume service.**

```
$ nova-manage sm flavor_create <label> <description>

$ nova-manage sm flavor_delete <label>

$ nova-manage sm backend_add <flavor label> <SR type> [config connection
 parameters]

Note: SR type and config connection parameters are in keeping with the
 XenAPI Command Line Interface. http://support.citrix.com/article/CTX124887

$ nova-manage sm backend_delete <backend-id>
```

Example: For the NFS storage manager plugin, the steps below may be used.

```
$ nova-manage sm flavor_create gold "Not all that glitters"

$ nova-manage sm flavor_delete gold

$ nova-manage sm backend_add gold nfs name_label=mybackend server=myserver
 serverpath=/local/scratch/myname

$ nova-manage sm backend_remove 1
```

- **Start nova-volume and nova-compute with the new configuration options.**

Creating and Accessing the volumes from VMs

Currently, the flavors have not been tied to the volume types API. As a result, we simply end up creating volumes in a "first fit" order on the given backends.

The standard euca-* or openstack API commands (such as volume extensions) should be used for creating, destroying, attaching, or detaching volumes.

Boot From Volume

The Compute service has preliminary support for booting an instance from a volume.

Creating a bootable volume

To 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.

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>:<typ
                        e>:<size(GB)>:<delete_on_terminate>.
```

The command arguments are:

`dev_name` — A device name where the volume will be attached in the system at /dev/*dev_name*. This value is typically `vda`.

`id` — The ID of the volume to boot from, as shown in the output of **nova volume-list**.

`type` — This is either `snap`, which means that the volume was created from a snapshot, or anything other than `snap` (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 `True` or `1`. False can be specified as `False` or `0`.

Note

Because of bug #1008622, you must specify an image when booting from a volume, even though this image will not be used.

The following example will attempt boot from volume with ID=13, it will not delete on terminate. Replace the `--image` flag with a valid image on your system, and the `--key_name` with a valid keypair name:

```
$ nova boot --image f4addd24-4e8a-46bb-b15d-fae2591f1a35 --flavor 2 --key_name
 mykey --block_device_mapping vda=13:::0 boot-from-vol-test
```

11. Scheduling

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.standard_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 **nova-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.

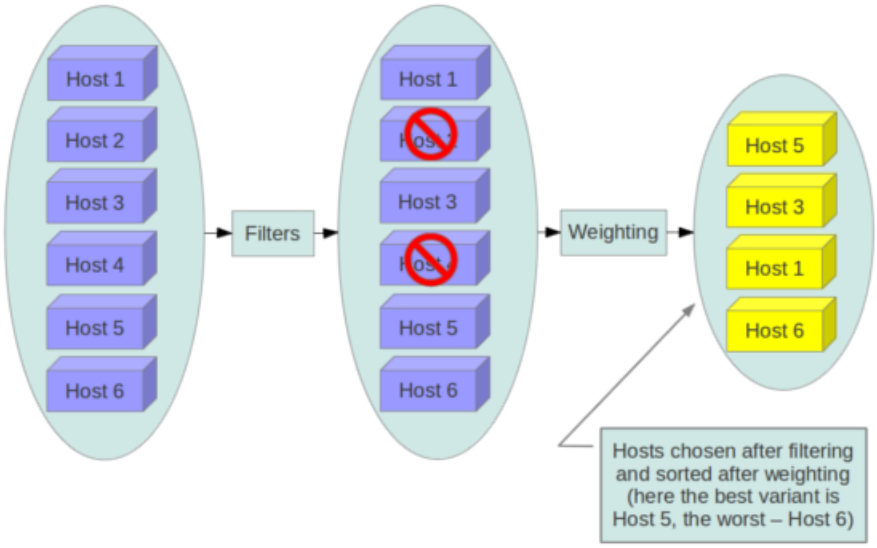
Filter Scheduler

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 11.1. 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 incldued with the Compute service:

```
scheduler_available_filters=nova.scheduler.filters.standard_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.standard_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 filtes are:

```
scheduler_default_filters=AvailabilityZoneFilter,RamFilter,ComputeFilter
```

The available filters are described below.

AllHostsFilter

This is a no-op filter, it does not eliminate any of the available hosts.

AvailabilityZoneFilter

Filters hosts by availabilty zone. This filter must be enabled for the scheduler to respect availability zones in requests.

ComputeFilter

Filters hosts by flavor (also known as instance type). The scheduler will check to ensure that a host has sufficient capabilities for the requested flavor. 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 overprovision 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 keyand 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-86d26cf818e1,8c19174f-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'],
    }
}
```

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 overprovision 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.

Warning

The `ram_allocation_ratio` option does not work properly in the Essex release, see bug #1016273 for details.

SameHostFilter

Schedule the instance on the same host as another instance in a set of 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-86d26cf818e1,8c19174f-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

Warning

The SimpleCIDRAffinityFilter does not work properly in the Essex release. See bug 999928 for details.

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`

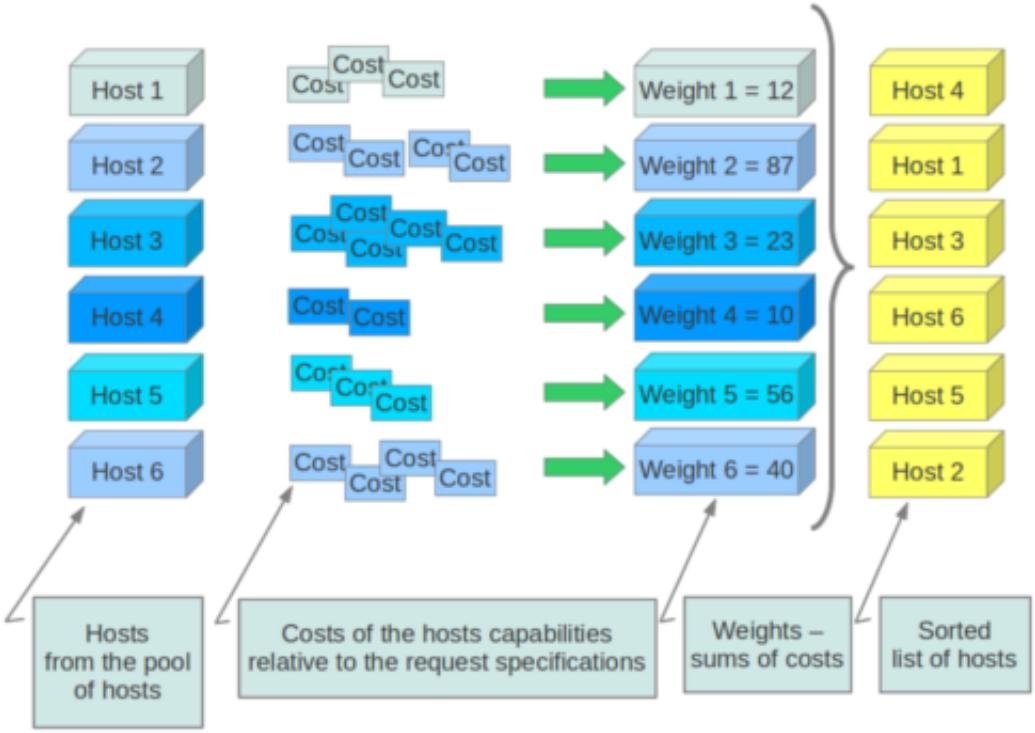
```
$ nova boot --image cedef40a-ed67-4d10-800e-17455edce175 --flavor 1 --hint
 build_near_host_ip=192.168.1.1 --hint cidr=/24 server-1
```

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 11.2. 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 two 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.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 `nova-volume` requests. It is the default top-level scheduler as specified by the `scheduler_driver` configuration option.

Simple Scheulder

The Simple Scheduler (`nova.scheduler.simple.SimpleScheduler`) implements a naive scheduler that tries to find the least loaded host (i.e., implements a "spread-first" algorithm). It can schedule requests for both `nova-compute` and `nova-volume`.

The Simple Scheduler supports the following configuration options:

Table 11.1. Description of Simple Scheduler configuration options

Configuration option=Default value	(Type) Description
`max_cores=16`	(IntOpt) The maximum number of instance cores to allow per host. Used when servicing compute requests to determine whether a host is a valid candidate for launching a VM instance.
`max_gigabytes=10000`	(IntOpt) Maximum number of volume gigabytes to allow per host. Used when servicing volume requests to determine whether a host is a valid candidate for creating a new volume.
`skip_isolated_core_check=true`	(BoolOpt) If true, allow overcommitting of vcpus on isolated hosts.

12. System Administration

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, nova-volume, 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.

For example, other components that need to deal with volumes in some way, should do so by calling methods on the VolumeManager instead of directly changing fields in the database. This allows us to keep all of the code relating to volumes in the same place.

- 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-volume - The nova-volume service is responsible for managing attachable block storage devices. It loads a Service object which exposes the public methods on VolumeManager via 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), volumes (block storage), 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

Volume Workers

Volume Workers interact with iSCSI storage to manage LVM-based instance volumes. Specific functions include:

- Create volumes
- Delete volumes
- Establish Compute volumes

Volumes may easily be transferred between instances, but may be attached to only a single instance at a time.

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-manage command, and the novaclient or the Euca2ools commands.

The nova-manage command may only be run by users with admin privileges. Both novaclient and euca2ools can be used by all users, though specific commands may be restricted by Role Based Access Control in the deprecated nova auth system or in the Identity Management service.

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.

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 user

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 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=joecool
$ 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 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

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-volume 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 nuber of cpu
  - **mem(mb):**total amount of memory (MB)
  - **hdd**total amount of 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 bee83dd3-5cc9-47bc-a1bd-6d11186692d0 HostC
Migration of bee83dd3-5cc9-47bc-a1bd-6d11186692d0 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.

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. While the Diablo release includes the snapshot functions, both the backup procedure and the utility do apply to the Cactus release.

For reference, you cand find a DRP definition here : 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-objecstore, nova-volume, nova-network)

2. A compute node (nova-compute)

3. A Storage Area Network used by nova-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 "nova-volumes" LVM's VG).

- From the cloud controller to the compute node we also have active iscsi sessions (managed by nova-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 acess 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**

- **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 nova;
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 reacheable, 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) 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 performe 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 nova-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 nova-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:

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

- For your instances, if you have the whole `/home/` directory on the disk, 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 dettach 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 :

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

Do not forget the flag -r; otherwise, you will close ALL sessions.

13. OpenStack Interfaces

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. For more information about the Openstack-Dashboard project, please visit: https://github.com/openstack/horizon/. 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.

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 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`. Open `/etc/openstack-dashboard/local_settings.py` 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`.

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.

Configuring the Dashboard

Start the mysql command line client by running:

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

Enter the MySQL root user's password when prompted.

To configure the MySQL database, create the dash database.

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

Create a MySQL user for the newly-created dash database that has full control of the database.

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

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

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

A full example `local_settings.py` file is included in the Appendix of the OpenStack Install and Deploy manual.

In the `/etc/openstack-dashboard/local_settings.py` file, change these options:

- DATABASES: Change the database section to point to the MySQL database named `dash`:

```
DATABASES = {
    'default': {
        'ENGINE': 'django.db.backends.mysql',
        'NAME': 'dash',
        'USER': 'dash',
        'PASSWORD': 'yourpassword',
        'HOST': 'localhost',
        'default-character-set': 'utf8'
    },
}
```

- `SWIFT_ENABLED`: If an Object Storage (Swift) endpoint is available and configured in the Identity service catalog, set `SWIFT_ENABLED = True`.

- `QUANTUM_ENABLED`: If a Network Connection (Quantum) service is available and configured in the Identity service catalog, set `QUANTUM_ENABLED = True`. The project is expected to become a core OpenStack project in the Folsom release. You can set also `QUANTUM_ENABLED = False`.

Run the **manage.py syncdb** command to initialize the database.

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

As a result, you should see the following at the end of what returns:

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

If you want to avoid a warning when restarting apache2, create a blackhole directory in the dashboard directory like so:

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

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

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

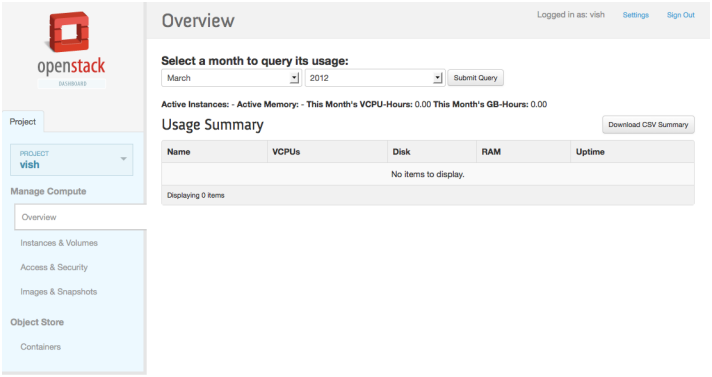
Restart the nova-api service to ensure the API server can connect to the Dashboard and to avoid an error displayed in the Dashboard.

```
sudo restart nova-api
```

Validating the Dashboard Install

To validate the Dashboard installation, point your browser at http://192.168.206.130. Note that you cannot use VNC Console from a Chrome browser. You need both Flash installed and a Firefox browser. 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 "adminUser" for the username and "secretword" as the password.

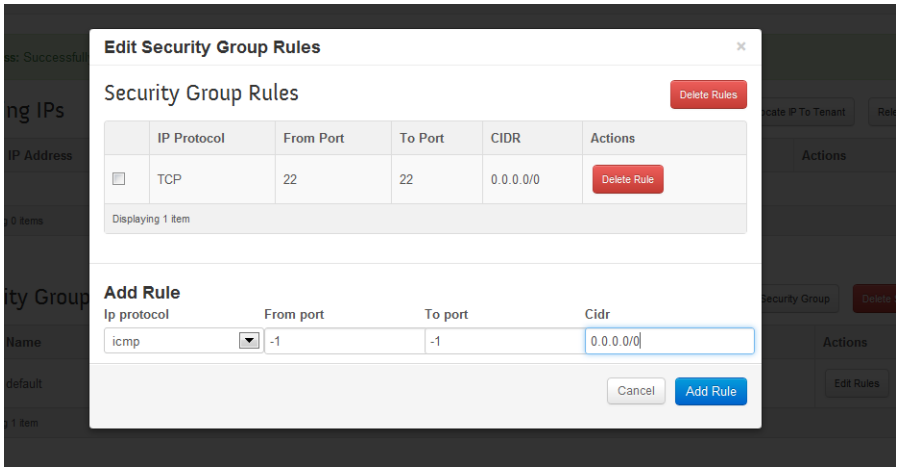


Launching Instances using Dashboard

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.

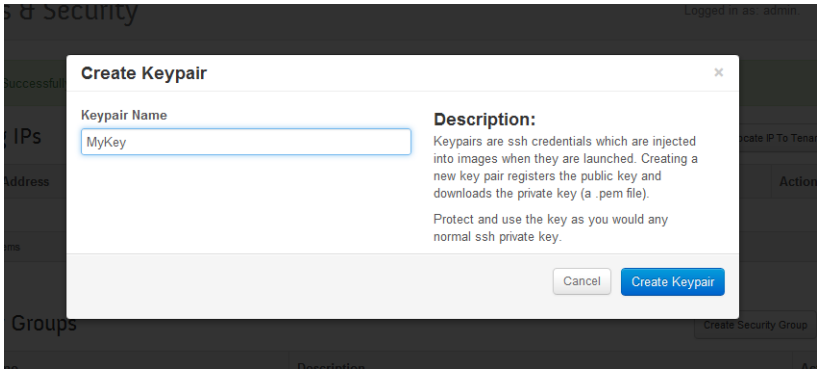


Select IP protocol TCP and enter 22 in "From Port" and "To Port" and CIDR 0.0.0.0/0. This opens port 22 for requests from any IP. If you want requests from particular range of IP, provide it in CIDR field.

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

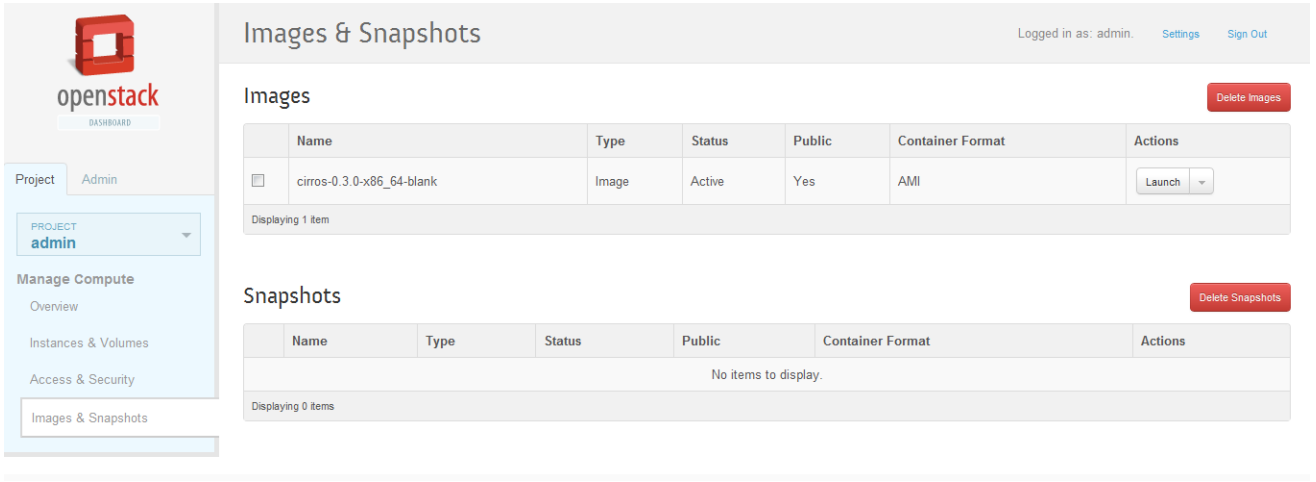
Next add a Keypair. Once a Keypair is added, the public key would be downloaded. This key can be used to SSH to the launched instance.



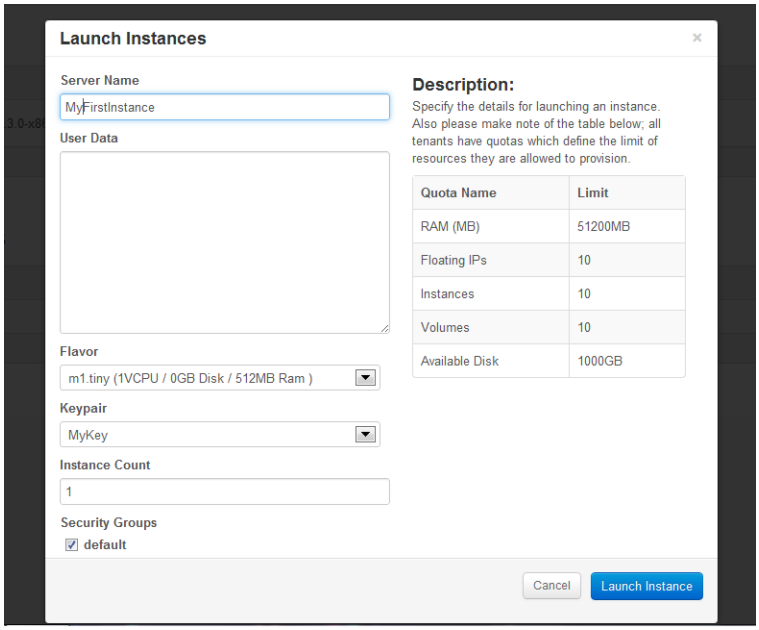
Once this is done, we are now all set to launch an Instance

Launching Instance

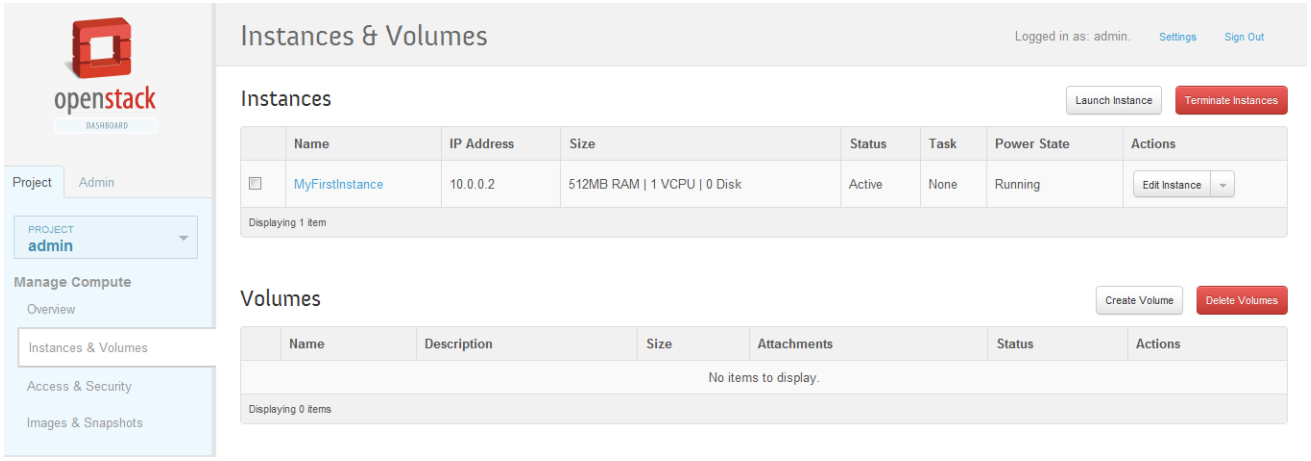
Click Images & Snapshots and launch a required instance from the list of images available.



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.



Once the status is Active, the instance is ready and we can ping and SSH to the instance.



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.

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. In Essex and beyond, there is support for for both libvirt and XenServer using both Java and websocket cleints.

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 what 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.

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 for either proxy to work. Many proxies of either type can be run against a single nova-consoleauth service in a cluster configuration.

nova-consoleauth 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 novaclient:

```
$ 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
```

Important nova-compute Options

To enable vncproxy in your cloud, in addition to running one or both of the proxies and nova-consoleauth, you need to configure the following options in `nova.conf` on your compute hosts.

- `[no]vnc_enabled` - Defaults to enabled. If this option is disabled your instances will launch without VNC support.
- `vncserver_listen` - Defaults to `127.0.0.1`. This is the address that vncservers will bind, and should be overridden in production deployments as a private address. Applies to libvirt only. For multi-host libvirt deployments this should be set to a host management IP on the same network as the proxies.

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. The result is that live migration will fail and the following error will appear in the `libvirtd.log` file in the destination host:

```
error: qemuMonitorIORead:513 : Unable to read from monitor:
 Connection reset by peer
```

If you wish to support live migration in your deployment, you must specify a value of `0.0.0.0` for `vncserver_listen`.

- `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 vncservers. 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.
- `novncproxy_base_url=[base url for client connections]` - This is the public base URL to which clients will connect. "?token=abc" will be added to this URL for the purposes of auth. When using the system as described in this document, an appropriate value is "http://$SERVICE_HOST:6080/vnc_auto.html" where *SERVICE_HOST* is a public hostname.
- `xvpvncproxy_base_url=`[base url for client connections] - This is the public base URL to which clients will connect. "?token=abc" will be added to this URL for the purposes of auth. When using the system as described in this document, an appropriate value is "http://$SERVICE_HOST:6081/console" where SERVICE_HOST is a public hostname.

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 `0.0.0.0:6080`. This can be configured in `nova.conf` with:

- `novncproxy_port=[port]`
- `novncproxy_host=[host]`

Note

Tthe previous vnc proxy implementation, called nova-vncproxy, has been deprecated

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 has changed since Diablo?

A: Previously, VNC support was done differently for libvirt and XenAPI. Now, there is unified multi-hypervisor support. To support this change, configuration options have been added and changed. Also, a new required service called nova-consoleauth has been added. If you are upgrading from Diablo, you will have to take these changes into consideration when upgrading.

If you are using Diablo, please see the documentation that shipped with your code, as this information will not be relevant.

Q: What happened to Diablo's nova-vncproxy?

A: nova-vncproxy was removed from the nova source tree. The Essex analog for this process is nova-novncproxy, which is provided by an external project.

Q: Why is nova-vncproxy no longer part of nova?

A: In Diablo, we shipped a websocket proxy (nova-vncproxy) with nova, but it had poor browser support. This nova-vncproxy code was dependent on external noVNC code,

so changes to that system involved updating 2 projects. Due to the rapid evolution of websocket tech, and the tight dependence of the websocket proxy on javscript and html components, we decided to keep that code all in one place.

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 novncproxy_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+). Also, if you are using Diablo's nova-vncproxy, note that support for this protocol is not provided.

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/templates/nova/instances_and_volumes/instances/_detail_vnc.html`.

Modify the `width` and `height` parameters:

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

14. OpenStack Compute Automated Installations

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 Essex(Nova with dashboard, Glance, Swift, Keystone)
- OpenStack Diablo(Nova, Glance, Swift)
- hadoop 0.20.2
- 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 14.1. OSes supported

	ubuntu 10.10	ubuntu 11.04	ubuntu 11.10	ubuntu 12.04
OpenStack Essex (Nova with dashboard, Glance, Swift, Keystone)				:)
OpenStack Diablo (Nova, Glance, Swift)	:)	:)	:)	
hadoop 0.20.2	:)	:)	:)	

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 node $server
   ```

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
```

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/`.

Using REST APIs

An API simulator can be found at `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 diablo,

```
$ sudo -i
$ cd /tmp/nova
$ . env/novarc
$ euca-describe-instances
$ ssh -i mykey.priv 10.0.0.3
```

```
```

For openstack nova essex,

```
$ 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.glance.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
   ```

15. OpenStack Compute Tutorials

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/ 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 diagnostics
```

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.

16. Support and Troubleshooting

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.

The Launchpad Answers area

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. One place to look for help is the Answers section on Launchpad. Launchpad is the "home" for the project code and its developers and thus is a natural place to ask about the project. When visiting the Answers section, 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. The Launchpad Answers areas are available here - OpenStack Compute: https://answers.launchpad.net/nova OpenStack Object Storage: https://answers.launchpad.net/swift.

OpenStack mailing list

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/.

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. "Austin release" vs lp:nova rev.396.
- Any deployment specific info is helpful as well. eg. Ubuntu 10.04, 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

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.

Troubleshooting OpenStack Object Storage

For OpenStack Object Storage, everything is logged in /var/log/syslog (or messages on some distros). Several settings enable further customization of logging, such as log_name, log_facility, and log_level, within the object server configuration files.

Handling Drive Failure

In the event that a drive has failed, the first step is to make sure the drive is unmounted. This will make it easier for OpenStack Object Storage to work around the failure until it has been resolved. If the drive is going to be replaced immediately, then it is just best to replace the drive, format it, remount it, and let replication fill it up.

If the drive can't be replaced immediately, then it is best to leave it unmounted, and remove the drive from the ring. This will allow all the replicas that were on that drive to be replicated elsewhere until the drive is replaced. Once the drive is replaced, it can be re-added to the ring.

Handling Server Failure

If a server is having hardware issues, it is a good idea to make sure the OpenStack Object Storage services are not running. This will allow OpenStack Object Storage to work around the failure while you troubleshoot.

If the server just needs a reboot, or a small amount of work that should only last a couple of hours, then it is probably best to let OpenStack Object Storage work around the failure and get the machine fixed and back online. When the machine comes back online, replication will make sure that anything that is missing during the downtime will get updated.

If the server has more serious issues, then it is probably best to remove all of the server's devices from the ring. Once the server has been repaired and is back online, the server's devices can be added back into the ring. It is important that the devices are reformatted before putting them back into the ring as it is likely to be responsible for a different set of partitions than before.

Detecting Failed Drives

It has been our experience that when a drive is about to fail, error messages will spew into /var/log/kern.log. There is a script called swift-drive-audit that can be run via cron to watch for bad drives. If errors are detected, it will unmount the bad drive, so that OpenStack Object Storage can work around it. The script takes a configuration file with the following settings:

```
            [drive-audit]
            Option  Default  Description
            log_facility  LOG_LOCAL0  Syslog log facility
            log_level  INFO  Log level
            device_dir  /srv/node  Directory devices are mounted under
            minutes  60  Number of minutes to look back in /var/log/kern.log
            error_limit  1  Number of errors to find before a device is
unmounted
```

This script has only been tested on Ubuntu 10.04, so if you are using a different distro or OS, some care should be taken before using in production.

Troubleshooting OpenStack Compute

Common problems for Compute typically involve misconfigured networking or credentials that are not sourced properly in the environment. Also, most flat networking configurations do not enable ping or ssh from a compute node to the instances 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 Launchpad Answers 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 to Launchpad Answers and IRC. 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.