OpenStack Compute

Administration Manual

openstack™

OpenStack Compute Administration Manual

trunk (2011-08-19)
Copyright © 2010, 2011 OpenStack LLC Some rights reserved.

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

Table of Contents

List of Figures

List of Tables

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/~openstack so 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 user'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.
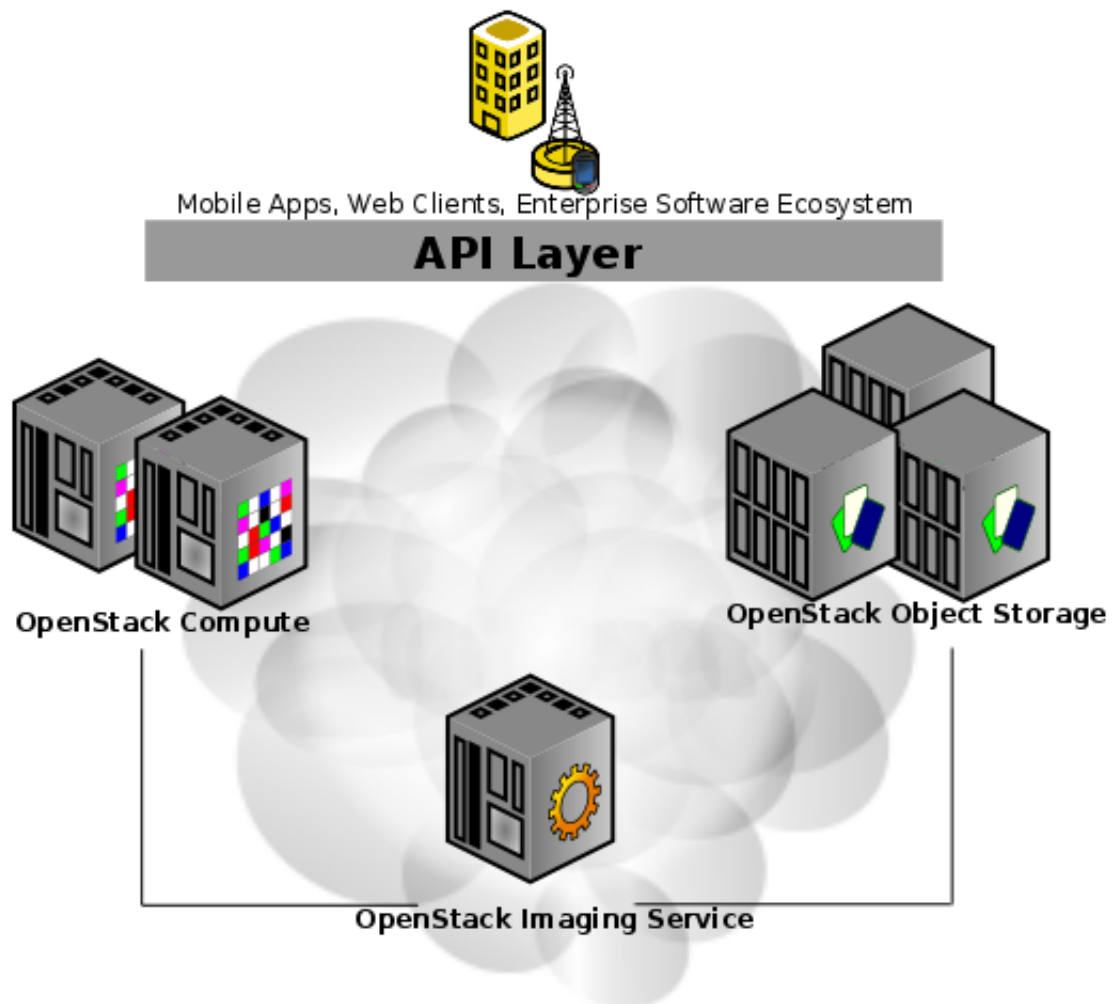
OpenStack Project Architecture Overview

by Ken Pepple

Before we dive into the conceptual and logic architecture, let's take a second to explain the OpenStack project:

> OpenStack is a collection of open source technologies delivering a massively scalable cloud operating system.

You can think of it as software to power your own Infrastructure as a Service (IaaS) offering like Amazon Web Services. It currently encompasses three main projects:

- Swift which provides object/blob storage. This is roughly analogous to Rackspace Cloud Files (from which it is derived) or Amazon S3.

- Glance which provides discovery, storage and retrieval of virtual machine images for OpenStack Nova.

- Nova which provides virtual servers upon demand. This is similar to Rackspace Cloud Servers or Amazon EC2.

While these three projects provide the core of the cloud infrastructure, OpenStack is open and evolving — there will be more projects (there are already related projects for web interfaces and a queue service). With that brief introduction, let's delve into a conceptual architecture and then examine how OpenStack Compute could map to it.

Cloud Provider Conceptual Architecture

Ken, Pepple

Imagine that we are going to build our own IaaS cloud and offer it to customers. To achieve this, we would need to provide several high level features:

1. Allow application owners to register for our cloud services, view their usage and see their bill (basic customer relations management functionality)

2. Allow Developers/DevOps folks to create and store custom images for their applications (basic build-time functionality)

3. Allow DevOps/Developers to launch, monitor and terminate instances (basic run-time functionality)

4. Allow the Cloud Operator to configure and operate the cloud infrastructure

While there are certainly many, many other features that we would need to offer (especially if we were to follow are more complete industry framework like eTOM), these four get to the very heart of providing IaaS. Now assuming that you agree with these four top level features, you might put together a conceptual architecture that looks something like this:
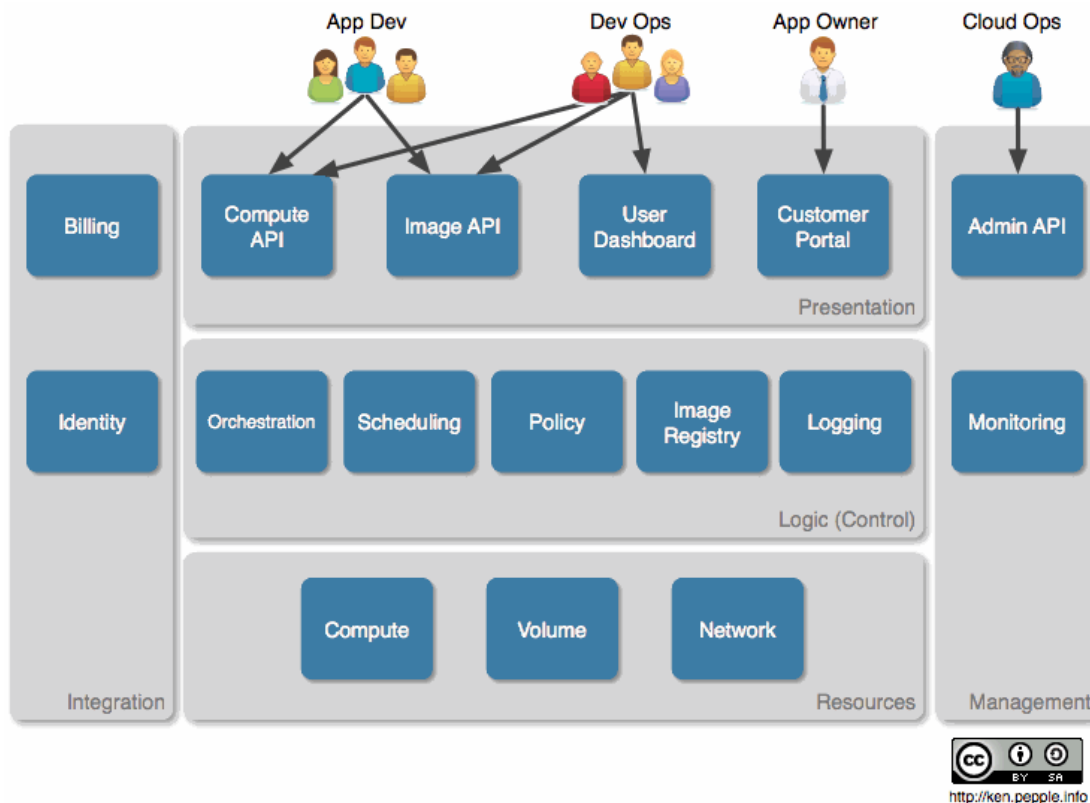
In this model, I've imagined four sets of users (developers, devops, owners and operators) that need to interact with the cloud and then separated out the functionality needed for each. From there, I've followed a pretty common tiered approach to the architecture (presentation, logic and resources) with two orthogonal areas (integration and management). Let's explore each a little further:

• As with presentation layers in more typical application architectures, components here interact with users to accept and present information. In this layer, you will find web portals to provide graphical interfaces for non-developers and API endpoints for developers. For more advanced architectures, you might find load balancing, console proxies, security and naming services present here also.

• The logic tier would provide the intelligence and control functionality for our cloud. This tier would house orchestration (workflow for complex tasks), scheduling (determining mapping of jobs to resources), policy (quotas and such) , image registry (metadata about instance images), logging (events and metering).

• There will need to integration functions within the architecture. It is assumed that most service providers will already have a customer identity and billing systems. Any cloud architecture would need to integrate with these systems.

• As with any complex environment, we will need a management tier to operate the environment. This should include an API to access the cloud administration features as well as some forms of monitoring. It is likely that the monitoring functionality will take the form of integration into an existing tool. While I've highlighted monitoring and an admin API for our fictional provider, in a more complete architecture you would

see a vast array of operational support functions like provisioning and configuration management.

- Finally, since this is a compute cloud, we will need actual compute, network and storage resources to provide to our customers. This tier provides these services, whether they be servers, network switches, network attached storage or other resources.

With this model in place, let's shift gears and look at OpenStack Compute's logical architecture.

OpenStack Compute Logical Architecture

Now that we've looked at a proposed conceptual architecture, let's see how OpenStack Compute is logically architected. At the time of this writing, Cactus was the newest release (which means if you are viewing this after around July 2011, this may be out of date). There are several logical components of OpenStack Compute architecture but the majority of these components are custom written python daemons of two varieties:

- WSGI applications to receive and mediate API calls (`nova-api`, `glance-api`, etc.)

- Worker daemons to carry out orchestration tasks (`nova-compute`, `nova-network`, `nova-schedule`, etc.)

However, there are two essential pieces of the logical architecture are neither custom written nor Python based: the messaging queue and the database. These two components facilitate the asynchronous orchestration of complex tasks through message passing and information sharing. Putting this all together we get a picture like this:
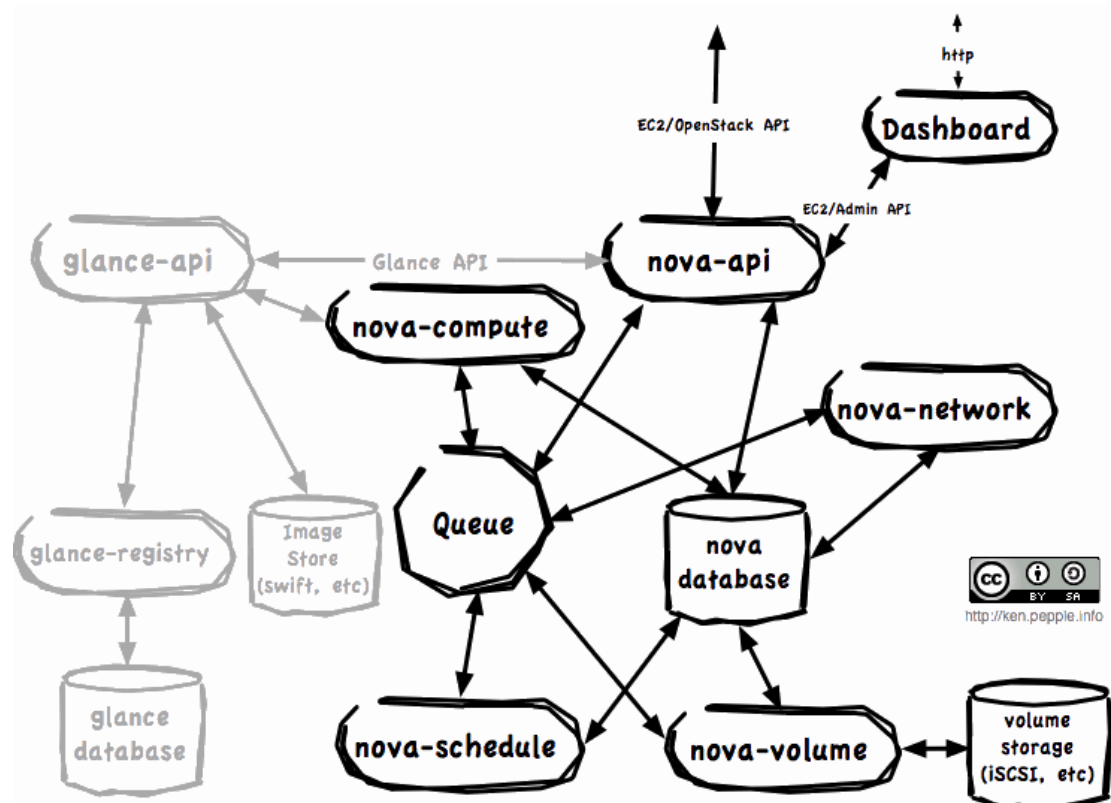
This complicated, but not overly informative, diagram as it can be summed up in three sentences:

- End users (DevOps, Developers and even other OpenStack components) talk to `nova-api` to interface with OpenStack Compute

- OpenStack Compute daemons exchange info through the queue (actions) and database (information) to carry out API requests

- OpenStack Glance is basically a completely separate infrastructure which OpenStack Compute interfaces through the Glance API

Now that we see the overview of the processes and their interactions, let's take a closer look at each component.

- The `nova-api` daemon is the heart of the OpenStack Compute. You may see it illustrated on many pictures of OpenStack Compute as API and "Cloud Controller". While this is partly true, cloud controller is really just a class (specifically the CloudController in trunk/nova/api/ec2/cloud.py) within the `nova-api` daemon. It provides an endpoint for all API queries (either OpenStack API or EC2 API), initiates most of the orchestration activities (such as running an instance) and also enforces some policy (mostly quota checks).

- The `nova-schedule` process is conceptually the simplest piece of code in OpenStack Compute: take a virtual machine instance request from the queue and determines where it should run (specifically, which compute server host it should run on). In practice however, I am sure this will grow to be the most complex as it needs to factor in current state of the entire cloud infrastructure and apply complicated algorithm to ensure efficient usage. To that end, `nova-schedule` implements a pluggable architecture that let's you choose (or write) your own algorithm for scheduling. Currently, there are several to choose from (simple, chance, etc) and it is a area of hot development for the future releases of OpenStack Compute.

- The `nova-compute` process is primarily a worker daemon that creates and terminates virtual machine instances. The process by which it does so is fairly complex (see this blog post by Laurence Luce for the gritty details) 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.

- As you can gather by the name, `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 AoE.

- 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 queue provides a central hub for passing messages between daemons. This is currently implemented with RabbitMQ today, but theoretically could be any AMPQ message queue supported by the python ampqlib.

- 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 Compute 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.

• OpenStack Glance is a separate project from OpenStack Compute, but as shown above, complimentary. While it is an optional part of the overall compute architecture, I can't imagine that most OpenStack Compute installations will not be using it (or a complimentary product). There are three pieces to Glance: `glance-api`, `glance-registry` and the image store. As you can probably guess, `glance-api` accepts API calls, much like `nova-api`, and the actual image blobs are placed in the image store. The `glance-registry` stores and retrieves metadata about images. The image store can be a number of different object stores, include OpenStack Object Storage (Swift).

• Finally, another optional project that we will need for our fictional service provider is an user dashboard. I have picked the OpenStack Dashboard here, but there are also several other web front ends available for OpenStack Compute. The OpenStack Dashboard provides a web interface into OpenStack Compute to give application developers and devops staff similar functionality to the API. It is currently implemented as a Django web application.

This logical architecture represents just one way to architect OpenStack Compute. With its pluggable architecture, we could easily swap out OpenStack Glance with another image service or use another dashboard. In the coming releases of OpenStack, expect to see more modularization of the code especially in the network and volume areas.

Nova Conceptual Mapping

Now that we've seen a conceptual architecture for a fictional cloud provider and examined the logical architecture of OpenStack Nova, it is fairly easy to map the OpenStack components to the conceptual areas to see what we are lacking:

http://ken.pepple.info

As you can see from the illustration, I've overlaid logical components of OpenStack Nova, Glance and Dashboard to denote functional coverage. For each of the overlays, I've added the name of the logical component within the project that provides the functionality. While all of these judgements are highly subjective, you can see that we have a majority coverage of the functional areas with a few notable exceptions:

• The largest gap in our functional coverage is logging and billing. At the moment, OpenStack Nova doesn't have a billing component that can mediate logging events, rate the logs and create/present bills. That being said, most service providers will already have one (or *many*) of these so the focus is really on the logging and integration with billing. This could be remedied in a variety of ways: augmentations of the code (which should happen in the next release "Diablo"), integration with commercial products or services (perhaps Zuora) or custom log parsing.

• Identity is also a point which will likely need to be augmented. Unless we are running a stock LDAP for our identity system, we will need to integrate our solution with OpenStack Compute. Having said that, this is true of almost all cloud solutions.

• The customer portal will also be an integration point. While OpenStack Compute provides a user dashboard (to see running instance, launch new instances, etc.), it doesn't provide an interface to allow application owners to signup for service, track their bills and lodge trouble tickets. Again, this is probably something that it is already in place at our imaginary service provider.

• Ideally, the Admin API would replicate all functionality that we'd be able to do via the command line interface (which in this case is mostly exposed through the nova-manage command). This will get better in the Diablo release with the Admin API work.

• Cloud monitoring and operations will be an important area of focus for our service provider. A key to any good operations approach is good tooling. While OpenStack Compute provides nova-instancemonitor, which tracks compute node utilization, we're really going to need a number of third party tools for monitoring.

• Policy is an extremely important area but very provider specific. Everything from quotas (which are supported) to quality of service (QoS) to privacy controls can fall under this. I've given OpenStack Nova partial coverage here, but that might vary depending on the intricacies of the providers needs. For the record, the Catus release of OpenStack Compute provides quotas for instances (number and cores used, volumes (size and number), floating IP addresses and metadata.

• Scheduling within OpenStack Compute is fairly rudimentary for larger installations today. The pluggable scheduler supports chance (random host assignment), simple (least loaded) and zone (random nodes within an availability zone). As within most areas on this list, this will be greatly augmented in Diablo. In development are distributed schedulers and schedulers that understand heterogeneous hosts (for support of GPUs and differing CPU architectures).

As you can see, OpenStack Compute provides a fair basis for our mythical service provider, as long as the mythical service providers are willing to do some integration here and there.

Note that since the time of this writing, OpenStack Identity Service has been added.

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 users'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:

- Hyper-V 2008

- 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 - XenServer 5.5, Xen Cloud Platform (XCP)

Users and Projects

The OpenStack Compute system is designed to be used by many different cloud computing consumers or customers, 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. 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. There are both global roles and per-project 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.

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. Projects 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 he or she wishes to use by appending :project_id to his or her access key. If no project is specified in the API request, Compute attempts to use a project with the same id as the user.

For projects, 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

An image is a file containing information about a virtual disk that completely replicates all information about a working computer at a point in time including operating system information and file system information. Compute can use certificate management for decrypting bundled images. For now, Compute relies on using the euca2ools command-line tools distributed by the Eucalyptus Team for adding, bundling, and deleting images.

There are two methods for managing images. Images can be served through the OpenStack Image Service, a project that is named Glance, or use the nova-objectstore service. With an OpenStack Image Service server in place, the Image Service fetches the image on to the host machine and then OpenStack Compute boots the image from the host machine. To place images into the service, you would use a ReST interface to stream them, and the service, in turn, streams that into a back end which could be S3,

OpenStack Object Storage (which can use an S3), or the local file system on the server where OpenStack Image Service is installed.

An instance is a running virtual machine within the cloud. An instance has a life cycle that is controlled by OpenStack Compute. Compute creates the instances and it is responsible for building a disk image, launching it, reporting the state, attaching persistent storage, and terminating it.

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. 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. 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 distributed data store. Updates to system state are written into this store, using atomic transactions when required. Requests for system state are read out of this store. In limited cases, the read results are cached within controllers for short periods of time (for example, the current list of system users.)

Storage and OpenStack Compute

A 'volume' is a detachable block storage device. You can think of it as a USB hard drive. It can only be attached to one instance at a time, so it does not work like a SAN. If you wish to expose the same volume to multiple instances, you will have to use an NFS or SAMBA share from an existing instance.

Every instance larger than m1.tiny starts with some local storage (up to 160GB for m1.xlarge). This storage is currently the second partition on the root drive.

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.

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.

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.
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 Xen-based hypervisors, the Xen wiki contains a list of compatible processors on the HVM Compatible Processors page. For XenServer-compatible Intel processors, refer to the Intel® Virtualization Technology List. For LXC, the VT extensions are not required.

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

Networking: 1000 Mbps are suggested. For OpenStack Compute, networking is configured on multi-node installations between the physical machines on a single subnet. For

networking between virtual machine instances, three network options are available: flat, DHCP, and VLAN. Two NICs (Network Interface Cards) are recommended on the server running nova-network.

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

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

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. The only co-dependency between possible multi-node installations is that the Dashboard must be installed nova-api server. 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.
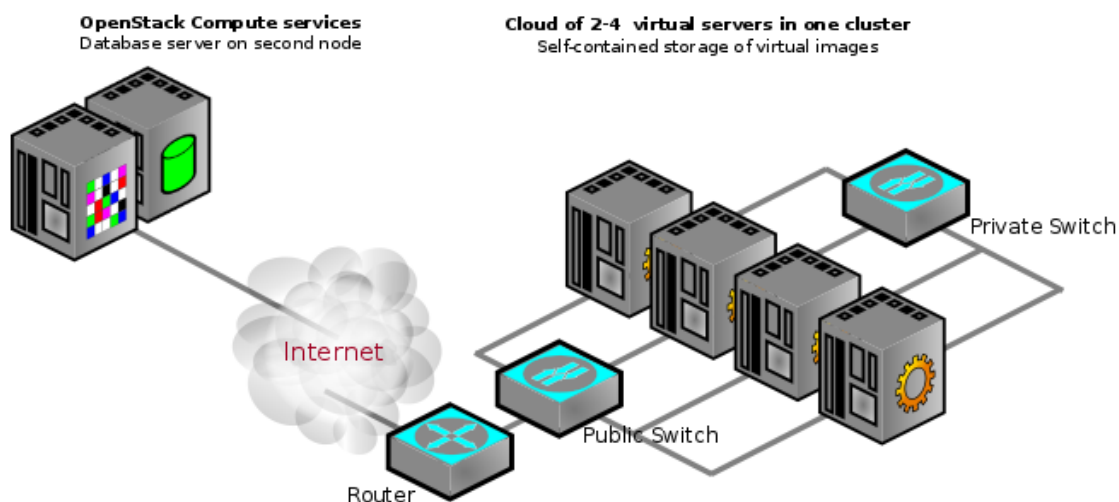
An alternative architecture would be to add more messaging servers if you notice a lot of back up in the messaging queue causing performance problems. In that case you would add an additional RabbitMQ server in addition to or instead of scaling up the database server. Your installation can run any nova- 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.
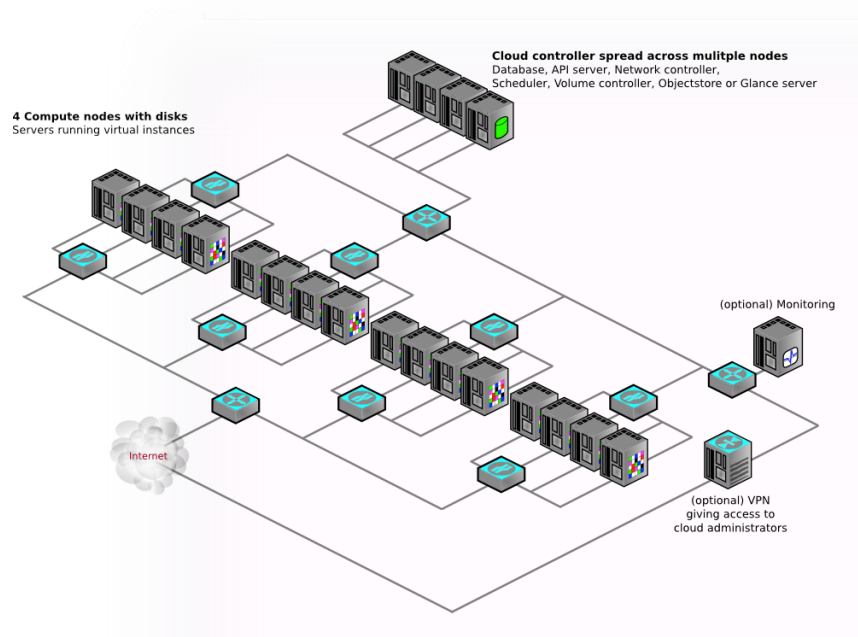
Service Architecture

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.
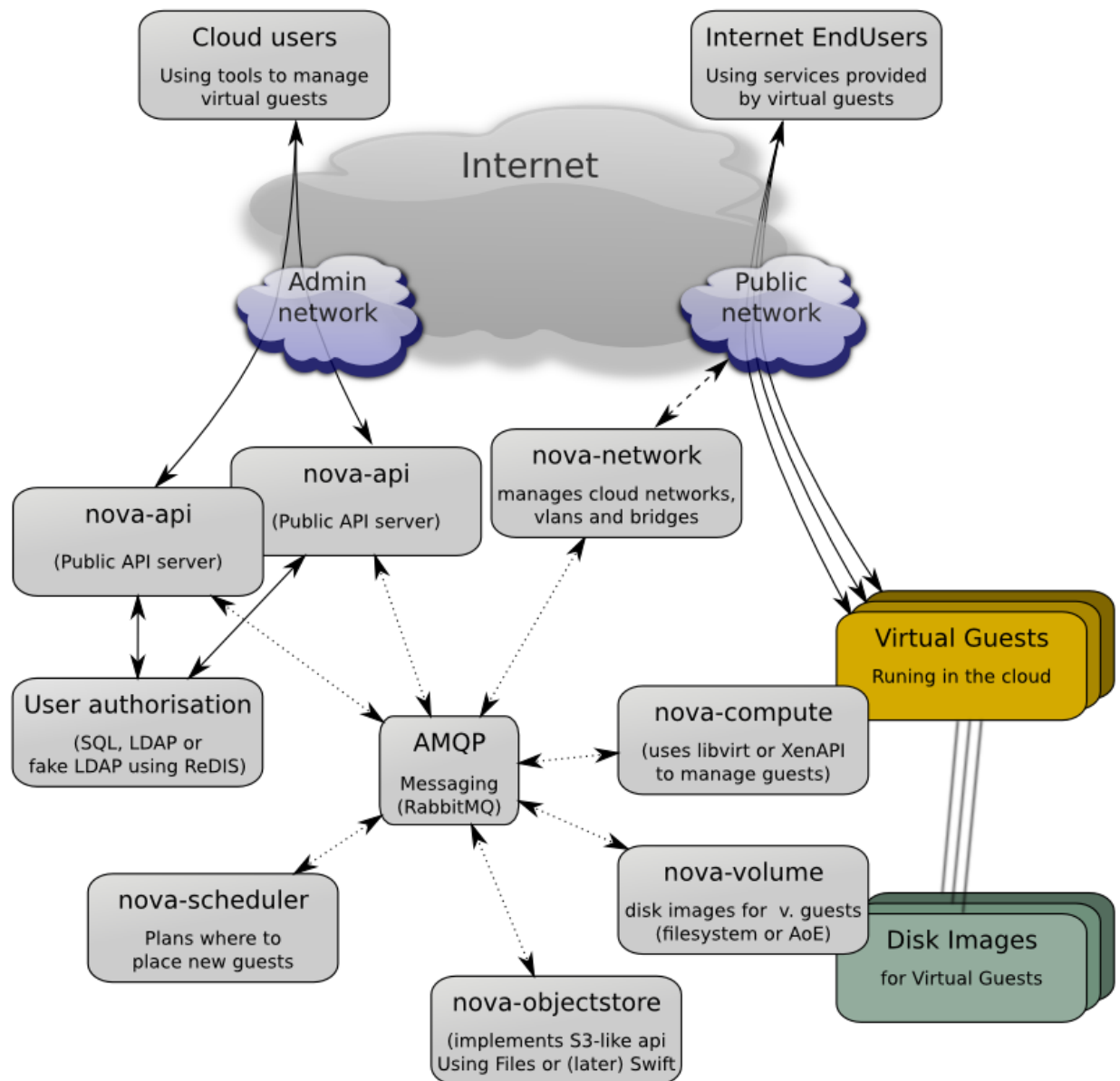
Installing OpenStack Compute on Ubuntu

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 from GitHub at https://github.com/elasticdog/OpenStack-NOVA-Installer-Script/raw/master/nova-install.

Copy the file to the servers where you want to install OpenStack Compute services - with multiple servers, you could install a cloud controller node and multiple compute nodes. The compute nodes manage the virtual machines through the nova-compute service. The cloud controller node contains all other nova- services.

Ensure you can execute the script by modifying the permissions on the script file.

```
wget --no-check-certificate https://github.com/elasticdog/OpenStack-NOVA-
Installer-Script/raw/master/nova-install
sudo chmod 755 nova-install
```

You must run the script with root permissions.

```
sudo bash nova-install -t cloud
```

The way this script is designed, you can have multiple servers for the cloud controller, the messaging service, and the database server, or run it all on one server. The -t or -type parameter has two options: `nova-install -t cloud` installs the cloud controller and `nova-install -t compute` installs a compute node for an existing cloud controller.

These are the parameters you enter using the script:

- Enter the Cloud Controller Host IP address.

- Enter the S3 IP, or use the default address as the current server's IP address.

- Enter the RabbitMQ Host IP. Again, you can use the default to install it to the local server. RabbitMQ will be installed.

- Enter the MySQL host IP address.

- Enter the MySQL root password and verify it.

- Enter a network range for all projects in CIDR format.

The script uses all these values entered for the configuration information to create the nova.conf configuration file. The script also walks you through creating a user and project. Enter a user name and project name when prompted. After the script is finished, you also need to create the project zip file. Credentials are generated after you create the project zip file with `nova-manage project zipfile projname username`

After configuring OpenStack Compute and creating a project zip file using the nova-manage project create command, be sure to unizp the project zip file and then source the novarc credential file that you extracted.

```
source /root/creds/novarc
```

Now all the necessary nova services are started up and you can begin to issue nova-manage commands. If you configured it to all run from one server, you're done. If you have a second server that you intend to use as a compute node (a node that does not contain the

database), install the nova services on the second node using the -t compute parameters using the same nova-install script.

To run from two or more servers, copy the nova.conf from the cloud controller node to the compute node.

Manual Installation

The manual installation involves installing from packages on Ubuntu 10.04 or 10.10 as a user with root permission. Depending on your environment, you may need to prefix these commands with sudo.

This installation process walks through installing a cloud controller node and a compute node. The cloud controller node contains all the nova- services including the API server and the database server. The compute node needs to run only the nova-compute service. You only need one nova-network service running in a multi-node install. You cannot install nova-objectstore on a different machine from nova-compute (production-style deployments will use a Glance server for virtual images).

Installing the Cloud Controller

First, set up pre-requisites to use the Nova PPA (Personal Packages Archive) provided through https://launchpad.net/~nova-core/+archive/trunk. The 'python-software-properties' package is a pre-requisite for setting up the nova package repository. You can also use the release package by adding the ppa:nova-core/release repository.

```
sudo apt-get install python-software-properties
```

```
sudo add-apt-repository ppa:nova-core/trunk
```

Run update.

```
sudo apt-get update
```

Install the messaging queue server, RabbitMQ.

```
sudo apt-get install -y rabbitmq-server
```

Now, install the Python dependencies.

```
sudo apt-get install -y python-greenlet python-mysqldb
```

Install the required nova- packages, and dependencies should be automatically installed.

```
sudo apt-get install -y nova-common nova-doc python-nova nova-api
                        nova-network nova-objectstore nova-scheduler nova-
compute
```

Install the supplemental tools such as euca2ools and unzip.

```
sudo apt-get install -y euca2ools unzip
```

Setting up the SQL Database (MySQL) on the Cloud Controller

You must use a SQLAlchemy-compatible database, such as MySQL or PostgreSQL. This example shows MySQL.

First you can set environments with a "pre-seed" line to bypass all the installation prompts,
running this as root:

```
bash
MYSQL_PASS=nova
NOVA_PASS=notnova
cat <<MYSQL_PRESEED | debconf-set-selections
mysql-server-5.1 mysql-server/root_password password $MYSQL_PASS
mysql-server-5.1 mysql-server/root_password_again password $MYSQL_PASS
mysql-server-5.1 mysql-server/start_on_boot boolean true
MYSQL_PRESEED
```

Next, install MySQL with: `sudo apt-get install -y mysql-server`

Edit /etc/mysql/my.cnf to change 'bind-address' from localhost (127.0.0.1) to any (0.0.0.0)
and restart the mysql service:

```
sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/my.cnf
sudo service mysql restart
```

To configure the MySQL database, create the nova database:

```
sudo mysql -uroot -p$MYSQL_PASS -e 'CREATE DATABASE nova;'
```

Update the DB to give user 'nova'@'%' full control of the nova database:

```
sudo mysql -uroot -p$MYSQL_PASS -e "GRANT ALL PRIVILEGES ON *.* TO
                      'nova'@'%' WITH GRANT OPTION;"
```

Set MySQL password for 'nova'@'%':

```
sudo mysql -uroot -p$MYSQL_PASS -e "SET PASSWORD FOR 'nova'@'%' =
                      PASSWORD('$NOVA_PASS');"
```

Installing the Compute Node

There are many different ways to perform a multinode install of Compute. In this case, you
can install all the nova- packages and dependencies as you did for the Cloud Controller
node, or just install nova-network and nova-compute. Your installation can run any nova-
services anywhere, so long as the service can access nova.conf so it knows where the
rabbitmq server is installed.

The Compute Node is where you configure the Compute network, the networking
between your instances. There are three options: flat, flatDHCP, and VLAN.

If you use FlatManager as your network manager, there are some additional networking
changes to ensure connectivity between your nodes and VMs. If you chose VlanManager or
FlatDHCP, you may skip this section because they are set up for you automatically.

Compute defaults to a bridge device named 'br100'. This needs to be created and
somehow integrated into your network. To keep things as simple as possible, have all the
VM guests on the same network as the VM hosts (the compute nodes). To do so, 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:

```
< begin /etc/network/interfaces >
```

OpenStack Compute
Administration Manual
            Aug 19, 2011
            trunk

```
# 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
< end /etc/network/interfaces >
```

Next, restart networking to apply the changes:

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

If you use flat networking, you must manually insert the IP address into the 'fixed_ips' table in the nova database. Also ensure that the database lists the bridge name correctly that matches the network configuration you are working within. Flat networking should insert this automatically but you may need to check it.

Because you may need to query the database from the Compute node and learn more information about instances, euca2ools and mysql-client packages should be installed on any additional Compute nodes.

Restart All Relevant Services on the Compute Node

On both nodes, restart all six services in total, just to cover the entire spectrum:

```
restart libvirt-bin; restart nova-network; restart nova-compute;
restart nova-api; restart nova-objectstore; restart nova-scheduler
```

Installing OpenStack Compute on Red Hat Enterprise Linux 6

This section documents a multi-node installation using RHEL 6. RPM repos for the Bexar release, the Cactus release, and also per-commit trunk builds for OpenStack Nova are available at http://yum.griddynamics.net.

Known limitations for RHEL version 6 installations:

• iSCSI LUN not supported due to tgtadm vs ietadm differences

• Only KVM hypervisor has been tested with this installation

To install Nova on RHEL v.6 you need access to two repositories, one available on the yum.griddynamics.net website and the RHEL DVD image connected as repo.

First, install RHEL 6.0, preferrably with a minimal set of packages.

Disable SELinux in /etc/sysconfig/selinux and then reboot.

Connect the RHEL 3. 6.0 x86_64 DVD as a repository in YUM.

```
sudo mount /dev/cdrom /mnt/cdrom
cat /etc/yum.repos.d/rhel.repo
[rhel]
name=RHEL 6.0
baseurl=file:///mnt/cdrom/Server
enabled=1
gpgcheck=0
```

Download and install repo config and key.

```
wget http://yum.griddynamics.net/openstack-repo-2011.1-2.noarch.rpm
sudo rpm -i openstack-repo-2011.1-2.noarch.rpm
```

Install the libvirt package (these instructions are tested only on KVM).

```
sudo yum install libvirt
sudo chkconfig libvirtd on
sudo service libvirtd start
```

Repeat the basic installation steps to put the pre-requisites on all cloud controller and compute nodes. Nova has many different possible configurations. You can install Nova services on separate servers as needed but these are the basic pre-reqs.

These are the basic packages to install for a cloud controller node:

```
sudo yum install euca2ools openstack-nova-
{api,compute,network,objectstore,scheduler,volume} openstack-nova-cc-config
 openstack-glance
```

These are the basic packages to install compute nodes. Repeat for each compute node (the node that runs the VMs) that you want to install.

```
sudo yum install openstack-nova-compute openstack-nova-compute-config
```

On the cloud controller node, create a MySQL database named nova.

```
sudo service mysqld start
sudo chkconfig mysqld on
sudo service rabbitmq-server start
sudo chkconfig rabbitmq-server on
mysqladmin -uroot password nova
```

You can use this script to create the database.

```
#!/bin/bash

DB_NAME=nova
DB_USER=nova
DB_PASS=nova
PWD=nova

CC_HOST="A.B.C.D" # IPv4 address
HOSTS='node1 node2 node3' # compute nodes list

mysqladmin -uroot -p$PWD -f drop nova
mysqladmin -uroot -p$PWD create nova

for h in $HOSTS localhost; do
        echo "GRANT ALL PRIVILEGES ON $DB_NAME.* TO '$DB_USER'@'$h' IDENTIFIED
 BY '$DB_PASS';" | mysql -uroot -p$DB_PASS mysql
done
```

```
echo "GRANT ALL PRIVILEGES ON $DB_NAME.* TO $DB_USER IDENTIFIED BY
 '$DB_PASS';" | mysql -uroot -p$DB_PASS mysql
echo "GRANT ALL PRIVILEGES ON $DB_NAME.* TO root IDENTIFIED BY '$DB_PASS';" |
 mysql -uroot -p$DB_PASS mysql
```

Now, ensure the database version matches the version of nova that you are installing:

```
nova-manage db sync
```

On each node, set up the configuration file in /etc/nova/nova.conf.

Start the Nova services after configuring and you then are running an OpenStack cloud!

```
for n in api compute network objectstore scheduler volume; do sudo service
 openstack-nova-$n start; done
sudo service openstack-glance start
for n in node1 node2 node3; do ssh $n sudo service openstack-nova-compute
 start; done
```

Post-Installation Configuration for OpenStack Compute

Configuring your Compute installation involves nova-manage commands plus editing the nova.conf file to ensure the correct flags are set. 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, and you will read about additional configuration information in a separate chapter as well.

Setting Flags in the nova.conf File

The configuration file nova.conf is installed in /etc/nova by default. You only need to do these steps when installing manually, the scripted installation above does this configuration during the installation. A default set of options are already configured in nova.conf when you install manually. The defaults are as follows:

```
--daemonize=1
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
```

Starting with the default file, you must define the following required items in /etc/nova/nova.conf. The flag variables 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 flag settings, see the output of running /bin/nova-api –help.

Table 3.2. Description of nova.conf flags (not comprehensive)

Flag	Description
–sql_connection	IP address; Location of OpenStack Compute SQL database
–s3_host	IP address; Location where OpenStack Compute is hosting the objectstore service, which will contain the virtual machine images and buckets
–rabbit_host	IP address; Location of OpenStack Compute SQL database
–ec2_api	IP address; Location where the nova-api service runs

Flag	Description
–verbose	Set to 1 to turn on; Optional but helpful during initial setup
–ec2_url	HTTP URL; Location to interface nova-api. Example: http://184.106.239.134:8773/services/Cloud
–network_manager	Configures how your controller will communicate with additional OpenStack Compute nodes and virtual machines. Options: • nova.network.manager.FlatManager Simple, non-VLAN networking • nova.network.manager.FlatDHCPManager Flat networking with DHCP • nova.network.manager.VlanManager VLAN networking with DHCP; This is the Default if no network manager is defined here in nova.conf.
–fixed_range	IP address/range; Network prefix for the IP network that all the projects for future VM guests reside on. Example: 192.168.0.0/12
–network_size	Number value; Number of addresses in each private subnet.

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.

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--verbose
--s3_host=184.106.239.134
--rabbit_host=184.106.239.134
--ec2_api=184.106.239.134
--ec2_url=http://184.106.239.134:8773/services/Cloud
--fixed_range=192.168.0.0/16
--network_size=8
--routing_source_ip=184.106.239.134
--sql_connection=mysql://nova:notnova@184.106.239.134/nova
```

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 contains your MySQL server's username and password.

```
chown -R root: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, and then set up a user and project:

```
/usr/bin/nova-manage db sync
/usr/bin/nova-manage user admin <user_name>
/usr/bin/nova-manage project create <project_name> <user_name>
/usr/bin/nova-manage network create <project-network> <number-of-networks-in-
project> <addresses-in-each-network>
```

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

```
/usr/bin/nova-manage db sync
/usr/bin/nova-manage user admin dub
/usr/bin/nova-manage project create dubproject dub
/usr/bin/nova-manage network create 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.o.

When you run the `nova-manage network create` command, entries are made in the 'networks' and 'fixed_ips' table. 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 Certifications

Generate the certifications as a zip file. These are the certs you will use to launch instances, bundle images, and all the other assorted API functions.

```
mkdir -p /root/creds
/usr/bin/python /usr/bin/nova-manage project zipfile $NOVA_PROJECT
 $NOVA_PROJECT_USER /root/creds/novacreds.zip
```

If you are using one of the Flat modes for networking, you may see a Warning message "No vpn data for project <project_name>" which you can safely ignore.

Unzip them in your home directory, and add them to your environment.

```
unzip /root/creds/novacreds.zip -d /root/creds/
cat /root/creds/novarc >> ~/.bashrc
source ~/.bashrc
```

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:

```
euca-authorize -P icmp -t -1:-1 default
euca-authorize -P tcp -p 22 default
```

Another common issue is you cannot ping or SSH 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:

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

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.'

With the Bexar release we have two configuration files: nova-api.conf and nova.conf. 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. Customize the nova.config example below to match your environment. The CC_ADDR is the Cloud Controller IP Address.

```
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--logdir=/var/log/nova
--state_path=/var/lib/nova
--verbose
--sql_connection=mysql://root:nova@CC_ADDR/nova
--s3_host=CC_ADDR
--rabbit_host=CC_ADDR
--ec2_api=CC_ADDR
--ec2_url=http://CC_ADDR:8773/services/Cloud
--network_manager=nova.network.manager.FlatManager
--fixed_range= network/CIDR
--network_size=number of addresses
```

By default, Nova sets 'br100' as the bridge device, and this is what needs to be done next. 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:

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

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

```
restart libvirt-bin; 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

In the Diablo release, you can find the version of the installation by using the nova-manage command:

```
nova-manage version list
```

Migrating from Cactus to Diablo

If you have an installation already installed and running, to migrate to Diablo you must update the installation first, then your database, then perhaps your images if you were already running images on Cactus in the nova-objectstore. If you were running images through Glance, your images should work automatically after an upgrade. You can also export your users for importing into the OpenStack Identity Service (Keystone). Here are the overall steps.

If your installation already pointed to ppa:nova-core/release, the release package has been updated from Cactus to Diablo so you can simply run:

```
apt-get update
apt-get upgrade
```

Next, update the database schema.

```
nova-manage db sync
```

Restart all the nova- services.

A separate command is available to migrate users from the deprecated auth system to the Identity Service.

```
nova-manage shell export
```

Within the Keystone project there is a keystone-import script that you can run to import these users.

Make sure that you can launch images. You can convert images that were previously stored in the nova object store using this command:

```
nova-manage image convert /var/lib/nova/images
```

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

General Compute Configuration Overview

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

You can use a particular flag file by using the –flagfile (nova.conf) parameter when running one of the nova- services. This inserts flag definitions from the given configuration file name, which may be useful for debugging or performance tuning. Here are some general purpose flags.

Table 4.1. Description of general purpose nova.conf flags

Flag	Default	Description
–my_ip	None	IP address; Calculated to contain the host IP address.
–host	None	String value; Calculated to contain the name of the node where the cloud controller is hosted
-?, –[no]help	None	Show this help.
–[no]helpshort	None	Show usage only for this module.
–[no]helpxml	None	Show this help, but with XML output instead of text

If you want to maintain the state of all the services, you can use the –state_path flag 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. Here are additional flags that apply to all nova- services.

Table 4.2. Description of nova.conf flags for all services

Flag	Default	Description
–state_path	'/Users/username/p/nova/nova/../'	Directory path; Top-level directory for maintaining nova's state.
–periodic_interval	default: '60'	Integer value; Seconds between running periodic tasks.
–report_interval	default: '10'	Integer value; Seconds between nodes reporting state to the data store.

Example nova.conf Configuration Files

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

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

From wikimedia.org, used with permission. Where you see parameters passed in, it's likely an IP address you need.

```
# configured using KVM, FlatDHCP, MySQL, Glance, LDAP, and optionally
 sheepdog, API is EC2
--verbose
--daemonize=1
--logdir=/var/log/nova
--state_path=/var/lib/nova
--lock_path=/var/lock/nova
--sql_connection=mysql://$nova_db_user:$nova_db_pass@$nova_db_host/
$nova_db_name
--image_service=nova.image.glance.GlanceImageService
--s3_host=$nova_glance_host
--glance_api_servers=$nova_glance_host
--rabbit_host=$nova_rabbit_host
--network_host=$nova_network_host
--ec2_url=http://$nova_api_host:8773/services/Cloud
--libvirt_type=kvm
--dhcpbridge=/usr/bin/nova-dhcpbridge
--flat_network_bridge=br100
--network_manager=nova.network.manager.FlatDHCPManager
--flat_interface=$nova_network_flat_interface
--public_interface=$nova_network_public_interface
--routing_source_ip=$nova_network_public_ip
--ajax_console_proxy_url=$nova_ajax_proxy_url
--volume_driver=nova.volume.driver.SheepdogDriver
--auth_driver=nova.auth.ldapdriver.LdapDriver
--ldap_url=ldap://$nova_ldap_host
--ldap_password=$nova_ldap_user_pass
--ldap_user_dn=$nova_ldap_user_dn
--ldap_user_unit=people
--ldap_user_subtree=ou=people,$nova_ldap_base_dn
--ldap_project_subtree=ou=groups,$nova_ldap_base_dn
--role_project_subtree=ou=groups,$nova_ldap_base_dn
--ldap_cloudadmin=cn=cloudadmins,ou=groups,$nova_ldap_base_dn
--ldap_itsec=cn=itsec,ou=groups,$nova_ldap_base_dn
--ldap_sysadmin=cn=sysadmins,$nova_ldap_base_dn
--ldap_netadmin=cn=netadmins,$nova_ldap_base_dn
--ldap_developer=cn=developers,$nova_ldap_base_dn
```

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

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

```
# configured using KVM, Flat, MySQL, and Glance, API is OpenStack (or EC2)
--daemonize=1
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=/usr/bin/nova-dhcpbridge
--flat_network_bridge=br100
```

```
--lock_path=/var/lock/nova
--logdir=/var/log/nova
--state_path=/var/lib/nova
--verbose
--network_manager=nova.network.manager.FlatManager
--sql_connection=mysql://$nova_db_user:$nova_db_pass@$nova_db_host/
$nova_db_name
--osapi_host=$nova_api_host
--rabbit_host=$rabbit_api_host
--ec2_host=$nova_api_host
--image_service=nova.image.glance.GlanceImageService
--glance_api_servers=$nova_glance_host
# first 3 octets of the network your volume service is on, substitute with
 real numbers
--iscsi_ip_prefix=nnn.nnn.nnn
```

Configuring Logging

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

Table 4.3. Description of nova.conf flags for logging

Flag	Default	Description
–logdir	'/var/logs/nova'	Directory path; Output to a per-service log file in the named directory.
–logfile	default: ''	File name; Output to named file.
–[no]use_syslog	default: 'false'	Output to syslog using their file naming system.
–default_log_levels	default: 'amqplib=WARN,sqlalchemy=WARN,eventlet.wsgi.server=WARN'	Pair of named loggers and level of messages to be logged; List of logger=LEVEL pairs
–verbose	default: 'false'	Set to 1 or true to turn on; Shows debug output - optional but helpful during initial setup.

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

Table 4.4. Description of nova.conf flags for customized log formats

Flag	Default	Description
–logging_context_format_string	default: '%(asctime)s %(levelname)s %(name)s [%(request_id)s %(user)s %(project)s] %(message)s'	The format string to use for log messages with additional context.
–logging_debug_format_suffix	default: 'from %(processName)s (pid=%(process)d) %(funcName)s %(pathname)s:%(lineno)d'	The data to append to the log format when level is DEBUG
–logging_default_format_string	default: '%(asctime)s %(levelname)s %(name)s [-] %(message)s'	The format string to use for log messages without context.
–logging_exception_prefix	default: '(%(name)s): TRACE: '	String value; Prefix each line of exception output with this format.

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.

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 flag 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 4.5. Description of nova.conf flags for configuring IPv6

Flag	Default	Description
--use_ipv6	default: 'false'	Set to 1 or true to turn on; Determines whether to use IPv6 network addresses
--flat_injected	default: 'false'	Cactus only:Indicates whether Compute (Nova) should use attempt to inject IPv6 network configuration information into the guest. It attempts to modify /etc/network/interfaces and currently only works on Debian-based systems.

Configuring Image Service and Storage for Compute

You can either use a local image storage system or install Glance for storing and retrieving images. After you have installed a Glance server, you can configure nova-compute to use Glance for image storage and retrieval. You must change the --image_service flag to 'nova.image.glance.GlanceImageService' in order to use Glance to store and retrieve images for OpenStack Compute.

Table 4.6. Description of nova.conf flags for the Glance image service and storage

Flag	Default	Description
--image_service	default: 'nova.image.local.LocalImageService'	The service to use for retrieving and searching for images. Images must be registered using euca2ools. Options: • nova.image.s3.S3ImageService S3 backend for the Image Service. • nova.image.local.LocalImageService Image service storing images to local disk. It assumes that image_ids are integers. This is the default setting if no image manager is defined here. • **nova.image.glance.GlanceImageService** Glance back end for storing and retrieving images; See http://glance.openstack.org for more info.
--glance_api_servers	default: '$my_ip:9292'	List of Glance API hosts. Each item may contain a host (or IP address) and port of an OpenStack Compute Image Service server (project's name is Glance)

Flag	Default	Description
--s3_dmz	default: '$my_ip'	IP address; For instances internal IP (a DMZ is shorthand for a demilitarized zone)
--s3_host	default: '$my_ip'	IP address: IP address of the S3 host for infrastructure. Location where OpenStack Compute is hosting the objectstore service, which will contain the virtual machine images and buckets.
--s3_port	default: '3333'	Integer value; Port where S3 host is running
--use_s3	default: 'true'	Set to 1 or true to turn on; Determines whether to get images from s3 or use a local copy

If you choose not to use Glance for the image service, you can use the object store that maintains images in a particular location, namely the state path on the server local to the nova.conf file. You can also use a set of S3 buckets to store images.

Table 4.7. Description of nova.conf flags for local image storage

Flag	Default	Description
--image_service	default: 'nova.image.local.LocalImageService'	The service to use for retrieving and searching for images. Images must be registered using euca2ools. Options: • nova.image.s3.S3ImageService S3 backend for the Image Service; In Cactus, the S3 image service wraps the other image services for use by the EC2 API. The EC2 api will always use the S3 image service by default so setting the flag is not necessary. • **nova.image.local.LocalImageService** Image service storing images to local disk. It assumes that image_ids are integers. • nova.image.glance.GlanceImageService Glance back end for storing and retrieving images; See http://glance.openstack.org for more info.
--state_path	'/Users/username/p/nova/nova/../'	Directory path; Top-level directory for maintaining nova's state.
--buckets_path	'$state_path/buckets'	Directory path; Directory established for S3-style buckets.
--images_path	'$state_path/images'	Directory path; Directory that stores images when using object store.

Configuring Live Migrations

The live migration feature is useful when you need to upgrade or installing patches to hypervisors/BIOS and you need the machines to keep running. For example, when one of HDD volumes RAID or one of bonded NICs is out of order. Also for regular periodic

maintenance, you may need to migrate VM instances. When many VM instances are running on a specific physical machine, you can redistribute the high load. Sometimes when VM instances are scattered, you can move VM instances to a physical machine to arrange them more logically.

Environments

- **OS:** Ubuntu 10.04/10.10 for both instances and host.

- **Shared storage:** NOVA-INST-DIR/instances/ has to be mounted by shared storage (tested using NFS).

- **Instances:** Instance can be migrated with ISCSI/AoE based volumes

- **Hypervisor:** KVM with libvirt

- **(NOTE1)** "NOVA-INST-DIR/instance" is expected that vm image is put on to. see "flags.instances_path" in nova.compute.manager for the default value

- **(NOTE2)** This feature is admin only, since nova-manage is necessary.

Sample Nova Installation before starting

- Prepare 3 servers at least, lets say, HostA, HostB and HostC

- nova-api/nova-network/nova-volume/nova-objectstore/ nova-scheduler(and other daemon) are running on HostA.

- nova-compute is running on both HostB and HostC.

- HostA export NOVA-INST-DIR/instances, HostB and HostC mount it.

- To avoid any confusion, NOVA-INST-DIR is same at HostA/HostB/HostC("NOVA-INST-DIR" shows top of install dir).

- HostA export NOVA-INST-DIR/instances, HostB and HostC mount it.

Pre-requisite configurations

1. Configure /etc/hosts, Make sure 3 Hosts can do name-resolution with each other. Ping with each other is better way to test.

   ```
   # ping HostA
   # ping HostB
   # ping HostC
   ```

2. Configure NFS at HostA by adding below to /etc/exports

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

   Change "255.255.0.0" appropriate netmask, which should include HostB/HostC. Then restart nfs server.

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

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

```
HostA:/ DIR nfs4 defaults 0 0
```

Then mount, check exported directory can be mounted.

```
# mount -a -v
```

If fail, try this at any hosts.

```
# iptables -F
```

Also, check file/daemon permissions. We expect any nova daemons are running as root.

```
# ps -ef | grep nova
root 5948 5904 9 11:29 pts/4 00:00:00 python /opt/nova-2010.4//bin/nova-api
root 5952 5908 6 11:29 pts/5 00:00:00 python /opt/nova-2010.4//bin/nova-
objectstore
... (snip)
```

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

```
# ls -ld NOVA-INST-DIR/instances/
drwxr-xr-x 2 root root 4096 2010-12-07 14:34 nova-install-dir/instances/
```

Same check at HostB and HostC

```
# ls -ld NOVA-INST-DIR/instances/
drwxr-xr-x 2 root root 4096 2010-12-07 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% /opt  ( <--- this line is
 important.)
```

4. 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"
```

then, restart libvirt. Make sure libvirt is restarted.

```
# stop libvirt-bin && start libvirt-bin
# ps -ef | grep libvirt
root 1145 1 0 Nov27 ? 00:00:03 /usr/sbin/libvirtd -d -l
```

5. Flag configuration. usually, you do not have to configure any flags. Below chart is only for advanced usage.

Table 4.8. Description of nova.conf flags for live migration

Flag	Default	Description
–live_migration_retry_count	default: 30	Retry count needed in live_migration. Sleep 1sec for each retry
–live_migration_uri	default: 'qemu+tcp://%s/system'	Define protocol used by live_migration feature. If you would like to use qemu+ssh, change this as described at http://libvirt.org/.
–live_migration_bandwidth	default: 0	Define bandwidth used by live migration.
–live_migration_flag	default: 'VIR_MIGRATE_UNDEFINE_SOURCE, VIR_MIGRATE_PEER2PEER'	Define libvirt flag for live migration.

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 flags for database access

Flag	Default	Description
–sql_connection	default: 'sqlite:///$state_path/nova.sqlite'	IP address; Location of OpenStack Compute SQL database

Flag	Default	Description
--sql_idle_timeout	default: '3600'	
--sql_max_retries	default: '12'	Integer value; Number of attempts on the SQL connection
--sql_retry_interval	default: '10'	Integer value; Retry interval for SQL connections
--db_backend	default: 'sqlalchemy'	The backend selected for the database connection
--db_driver	default: 'nova.db.api'	The drive to use for database access

Configuring the Compute Messaging System

OpenStack Compute uses an open standard for messaging middleware known as AMQP. RabbitMQ enables this messaging system so that nova- services can talk to each other. 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 flags for Remote Procedure Calls and RabbitMQ Messaging

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

Table 4.11. Description of nova.conf flags for Tuning RabbitMQ Messaging

Flag	Default	Description
--rabbit_max_retries	default: '12'	Integer value; RabbitMQ connection attempts.
--rabbit-retry-interval	default: '10'	Integer value: RabbitMQ connection retry interval.
--rpc_thread_pool_size	default: '1024'	Integer value: Size of Remote Procedure Call thread pool.

Table 4.12. Description of nova.conf flags for Customizing Exchange or Topic Names

Flag	Default	Description
--control_exchange	default:nova	String value; Name of the main exchange to connect to
--ajax_console_proxy_topic	default: 'ajax_proxy'	String value; Topic that the ajax proxy nodes listen on
--console_topic	default: 'console'	String value; The topic console proxy nodes listen on

Flag	Default	Description
–network_topic	default: 'network'	String value; The topic network nodes listen on.
–scheduler_topic	default: 'scheduler'	String value; The topic scheduler nodes listen on.
–volume_topic	default: 'volume'	String value; Name of the topic that volume nodes listen on

Configuring Authentication and Authorization

There are different methods of authentication for the OpenStack Compute project and the default setting is now to use the OpenStack Identity Service, code-named Keystone. For the older, deprecated auth system, you set the –use_deprecated-auth flag. For no auth, use the default paste.ini that is included in the etch directory.

OpenStack Compute uses an implementation of an authentication system structured like having an Active Directory or other federated LDAP user store that backends to an identity manager or other SAML Policy Controller that then maps to groups. Credentials for API calls are stored in the project zip file when using the deprecated auth system. Certificate authority is also customized in nova.conf for the deprecated auth system.

If you see errors such as "EC2ResponseError: 403 Forbidden" it is likely you are trying to use euca commands without the auth system properly configured. Either install and configure the Identity Service, use the deprecated auth setting, or change out the default paste.ini file to use no auth.

Table 4.13. Description of nova.conf flags for Authentication

Flag	Default	Description
–auth_driver	default:'nova.auth.dbdriver.DbDriver'	String value; Name of the driver for authentication • nova.auth.dbdriver.DbDriver - Default setting, uses Identity Service (Keystone). • nova.auth.ldapdriver.FakeLdapDriver - create a replacement for this driver supporting other backends by creating another class that exposes the same public methods.
–use_deprecated_auth	default:'false'	True or false; Sets the auth system to use the zip file provided with the project files to store all credentials

Table 4.14. Description of nova.conf flags for customizing roles in deprecated auth

Flag	Default	Description
–allowed_roles	default: 'cloudadmin,itsec,sysadmin,netadmin,developer')	Comma separated list; Allowed roles for project
–global_roles	default: 'cloudadmin,itsec')	Comma separated list; Roles that apply to all projects
–superuser_roles	default: 'cloudadmin')	Comma separated list; Roles that ignore authorization checking completely

Table 4.15. Description of nova.conf flags for credentials in deprecated auth

Flag	Default	Description
–credentials_template	default: '')	Directory; Template for creating users' RC file
–credential_rc_file	default: '%src')	File name; File name of rc in credentials zip
–credential_cert_file	default: 'cert.pem')	File name; File name of certificate in credentials zip
–credential_key_file	default: 'pk.pem')	File name; File name of rc in credentials zip
–vpn_client_template	default: 'nova/cloudpipe/client/ovpn.template')	Directory; Refers to where the template lives for creating users vpn file
–credential_vpn_file	default: 'nova-vpn.conf')	File name; Filename of certificate in credentials.zip

Table 4.16. Description of nova.conf flags for CA (Certificate Authority)

Flag	Default	Description
–keys_path	default: '$state_path/keys')	Directory; Where Nova keeps the keys
–ca_file	default: 'cacert.pem')	File name; File name of root CA
–crl_file	default: 'crl.pem')	File name; File name of Certificate Revocation List
–key_file	default: 'private/cakey.pem')	File name; File name of private key
–use_project_ca	default: 'false')	True or false; Indicates whether to use a CA for each project; false means CA is not used for each project
–project_cert_subject	default: '/C=US/ST=California/ L=MountainView/O=AnsoLabs/ OU=NovaDev/CN=proje ct-ca-%s-%s')	String; Subject for certificate for projects, %s for project, timestamp
–user_cert_subject	default: '/C=US/ST=California/ L=MountainView/O=AnsoLabs/ OU=NovaDev/CN=%s-%s-%s')	String; Subject for certificate for users, %s for project, users, timestamp
–vpn_cert_subject	default: '/C=US/ST=California/ L=MountainView/O=AnsoLabs/ OU=NovaDev/CN=project-vpn-%s-%s')	String; Subject for certificate for vpns, %s for project, timestamp

5. Quick Guide to Getting Started with Keystone

The OpenStack Identity Service provides services for authenticating and managing user, account, and role information for OpenStack clouds running on OpenStack Compute and as an authorization service for OpenStack Object Storage.

Identity Service Concepts

The Keystone Identity Service has several key concepts which are important to understand:

User

A digital representation of a person, system, or service who uses OpenStack cloud services. Keystone 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 Keystone, authentication is the act of confirming the identity of a user or the truth of a claim. Keystone 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, Keystone 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 Keystone 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 Keystone, 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.

Installing the OpenStack Identity Service

You can install the Identity service from packages or from source.

To install the latest version of the Identity Service (Keystone) from the Github repositories, following the following instructions.

For Debian/Ubuntu, add the Keystone PPA to your sources.lst:

1. $> sudo add-apt-repository ppa:keystone-core/trunk $> sudo apt-get update

2. Install Keystone:

   $> sudo apt-get install keystone

To install the latest version of Keystone from the Launchpad Bazaar repositories, following the following instructions.

1. Grab the source tarball from Github

2. Untar the source tarball:

   $> tar -xzf <FILE>

3. Change into the package directory and build/install:

   $> cd keystone-<RELEASE> $> sudo python setup.py install

To install the latest version of Keystone from the Github repositories, see the following instructions.

These are for Debian/Ubuntu.

1. Install Git and build dependencies:

   $> sudo apt-get install git python-eventlet python-routes python-greenlet swift $> sudo apt-get install python-argparse python-sqlalchemy python-wsgiref python-pastedeploy

   ### Note

   If you want to build the Keystone documentation locally, you will also want to install the python-sphinx package.

1. Branch Keystone's trunk branch. (See http://wiki.openstack.org/GerritWorkflow to get the project initially setup):

   $> git checkout master $> git pull origin master

2. Install Keystone:

   $> sudo python setup.py install

Starting the Identity Service

To start up the Keystone service, enter the following:

```
$ cd ~/keystone/bin && ./keystone
```

In return you should see something like this:

```
Starting the Legacy Authentication component
        Service API listening on 0.0.0.0:5000
        Admin API listening on 0.0.0.0:5001
```

Configuring the Identity Service

Here are the steps to get started with authentication using Keystone, the project name for the OpenStack Identity Service.

Dependencies

Once Keystone is installed you need to initialize the database. You can do so with the keystone-manage command line utility. The keystone-manage utility helps with managing and configuring a Keystone installation. You configure the keystone-manage utility itself with a SQL Alchemy connection configuration via a parameter passed to the utility:

--sql_connection=CONN_STRING

Where the CONN_STRING is a proper SQLAlchemy connection string as described in http://www.sqlalchemy.org/docs/05/reference/sqlalchemy/connections.html?highlight=engine#sqlalchemy.create_engine.

One important use of keystone-manage is to setup the database. To do so, run:

```
keystone-manage db_sync
```

Creating Tenants, Users, Roles, Tokens and Endpoints

Sample data entries are available in keystone/bin/sampledata.sh. The following are just examples for a walk-through.

Note

Some reserved roles are defined (and can be modified) through the keystone.conf in the /etc folder.

Add two tenants, and administrative tenant and a tenant named demo. Tenants are equivalent to projects in the previous auth system in Compute. In Object Storage, Tenants are similar to accounts in the swauth system.

```
bin/keystone-manage  tenant add admin
  bin/keystone-manage  tenant add demo
```

Next add two users to the Identity Service and assign their passwords. The last value in the list is an ID number.

```
bin/keystone-manage  user add admin p4ssw0rd 1
  bin/keystone-manage  user add demo p455w0rd 2
```

Now you can assign roles, which includes a set of rights and privileges that are double-checked with the token that the user is issued.

```
bin/keystone-manage  role add Admin
  bin/keystone-manage  role add Member
  bin/keystone-manage  role grant Admin admin
```

Now define the endpointTemplates, which are URLs plus port values that indicate where a service may be accessed. This example shows many services available to Compute including the Image Service, the Object Storage service, as well as Identity itself. Since there is just one zone in this example, it represents all the services across the single region (but could also represent all the regions).

```
bin/keystone-manage  endpointTemplates add RegionOne swift http://%HOST_IP
%:8080/v1/AUTH_%tenant_id% http://%HOST_IP%:8080/ http://%HOST_IP%:8080/v1/
AUTH_%tenant_id% 1 1
  bin/keystone-manage  endpointTemplates add RegionOne nova_compat http://
%HOST_IP%:8774/v1.0/ http://%HOST_IP%:8774/v1.0  http://%HOST_IP%:8774/v1.0 1
 1
  bin/keystone-manage  endpointTemplates add RegionOne nova http://%HOST_IP
%:8774/v1.1/%tenant_id% http://%HOST_IP%:8774/v1.1/%tenant_id%  http://
%HOST_IP%:8774/v1.1/%tenant_id% 1 1
  bin/keystone-manage  endpointTemplates add RegionOne glance http://%HOST_IP
%:9292/v1.1/%tenant_id% http://%HOST_IP%:9292/v1.1/%tenant_id% http://%HOST_IP
%:9292/v1.1/%tenant_id% 1 1
  bin/keystone-manage  endpointTemplates add RegionOne identity http://
%HOST_IP%:5000/v2.0 http://%HOST_IP%:5001/v2.0 http://%HOST_IP%:5000/v2.0 1 1
```

Now you add a default token for the admin user to get when requesting a token.

```
bin/keystone-manage  token add 999888777666 1 1 2015-02-05T00:00
```

This section adds the tenant endpoints for each user created above (admin with ID 1 and demo with ID 2).

```
bin/keystone-manage  endpoint add 1 1
  bin/keystone-manage  endpoint add 1 2
  bin/keystone-manage  endpoint add 1 3
  bin/keystone-manage  endpoint add 1 4
  bin/keystone-manage  endpoint add 1 5
  bin/keystone-manage  endpoint add 1 6

  bin/keystone-manage  endpoint add 2 1
  bin/keystone-manage  endpoint add 2 2
  bin/keystone-manage  endpoint add 2 3
  bin/keystone-manage  endpoint add 2 4
  bin/keystone-manage  endpoint add 2 5
  bin/keystone-manage  endpoint add 2 6
```

You can configure Identity and Compute with a single region or multiple regions using zones. You need to add a label for the endpoint for each region. Having a single region doesn't require any work other than adding label.

```
keystone-manage endpointTemplates add SWRegion identity http://%HOST_IP%:5000/
v2.0 http://%HOST_IP%:5001/v2.0 http://%HOST_IP%:5000/v2.0 1 1
```

Curl examples

All examples assume default port usage (5001) and use the example admin account created above.

Admin Initial GET

Retrieves version, full API url, pdf doc link, and wadl link:

```
$> curl http://0.0.0.0:5001
```

or:

```
$> curl http://0.0.0.0:5001/v2.0/
```

Retrieve token:

To retrieve the token and expiration date for a user:

```
$> curl -d '{"passwordCredentials":{"username": "MyAdmin", "password":
 "P@ssw0rd"}}' -H "Content-type: application/json" http://localhost:5001/v2.0/
tokens
```

This will return something like:

```
$> {"auth": {"token": {"expires": "2011-08-10T17:45:22.838440", "id":
 "0eed0ced-4667-4221-a0b2-24c91f242b0b"}}}
```

Note

Save the "id" value as you'll be using it in the calls below.

To retrieve a list of tenants:

Run:

```
$> curl -H "X-Auth-Token:999888777666" http://localhost:5001/v2.0/tenants
```

This will return something like:

```
$> {"tenants": {"values": [{"enabled": 1, "id": "MyTenant", "description":
 null}], "links": []}}
```

Retrieve a list of users:

Run:

```
$> curl -H "X-Auth-Token:999888777666" http://localhost:5001/v2.0/users
```

This will return something like:

```
$> {"users": {"values": [{"email": null, "enabled": true, "id": "MyAdmin",
 "tenantId": "MyTenant"}], "links": []}}
```

Retrieve information about the token:

Run:

```
$> curl -H "X-Auth-Token:999888777666" http://localhost:5001/v2.0/tokens/
0eed0ced-4667-4221-a0b2-24c91f242b0b
```

This will return something like:

```
$> {"auth": {"token": {"expires": "2011-08-11T04:26:58.145171", "id":
 "0eed0ced-4667-4221-a0b2-24c91f242b0b"}, "user": {"username": "MyAdmin",
 "roleRefs": [{"roleId": "Admin", "id": 1}], "tenantId": "MyTenant"}}}
```

Revoking a token:

Run:

```
$> curl -X DELETE -H "X-Auth-Token:999888777666" http://localhost:5001/tokens/
0eed0ced-4667-4221-a0b2-24c91f242b0b
```

Creating a tenant:

Run:

```
 $> curl -H "X-Auth-Token:999888777666" -H "Content-type: application/
json" -d '{"tenant":{"id":"MyTenant2", "description":"My 2nd Tenant",
 "enabled":true}}'  http://localhost:5001/tenants
```

This will return something like:

```
$> {"tenant": {"enabled": true, "id": "MyTenant2", "description": "My 2nd
 Tenant"}}
```

Verifying the tenant:

Run:

```
$> curl -H "X-Auth-Token:999888777666" http://localhost:5001/v2.0/tenants/
MyTenant2
```

This will return something like:

```
$> {"tenant": {"enabled": 1, "id": "MyTenant2", "description": "My 2nd
 Tenant"}}
```

Updating the tenant:

Run:

```
$> curl -X PUT -H "X-Auth-Token:999888777666" -H "Content-type: application/
json" -d '{"tenant":{"description":"My NEW 2nd Tenant"}}' http://
localhost:5001/v2.0/tenants/MyTenant2
```

This will return something like:

```
$> {"tenant": {"enabled": true, "id": "MyTenant2", "description": "My NEW 2nd
 Tenant"}}
```

Deleting the tenant:

Run:

```
$> curl -X DELETE -H "X-Auth-Token:999888777666" http://localhost:5001/v2.0/
tenants/MyTenant2
```

6. 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:

- Hyper-V 2008 - Use to run Windows-based virtual machines, specifically Windows 2008 R2 Datacenter or Enterprise Edition. You must install and run nova-compute on Windows servers that run the Windows-based virtual machines.

- 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 5.5, Xen Cloud Platform (XCP), use to run Linux or Windows virtual machines. You must install the nova-compute service on DomU.

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 flag in nova.conf and restart the nova-compute service.

Here are the nova.conf flags that are used to configure the compute node.

Table 6.1. Description of nova.conf flags for the compute node

Flag	Default	Description
–connection_type	default: 'libvirt'	libvirt, xenapi, or fake; Value that indicates the virtualization connection type
–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
–use_cow_images	default: 'true'	True or false; Indicates whether to use cow images
–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

7. 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

Thanks to a new project available that couples Puppet automation with a configuration file and deployment tool, you can install many servers automatically by simply editing the configuration file (deploy.conf) and running the deployment tool (deploy.py in the nova-deployment-tool project in Launchpad).

Prerequisites

* Networking: The servers must be connected to a subnet.

* Networking: Ensure that the puppet server can access nova component servers by name. The command examples in this document identify the user as "nii". You should change the name but you need to create the same users on all Nova, Glance and Swift component servers in ~/DeploymentTool/conf/deploy.conf (ssh_user='user').

* Permissions: You must have root user permission for installation and service provision.

* Software: You must configure the installation server to access the Puppet server by name. (Puppet 0.25 or higher)

* Software: You must configure LVM if you do not change the default setting of the VolumeManager in the nova-volume service.

* Software: Python 2.6 or higher

* Software: Because of the current Nova implementation architecture, the binaries for nova-api, nova-objectstore, and euca2ools must have been loaded in one server.

* Operating system: Ubuntu 10.04, 10.10 or 11.04

The tool does not support system configurations other than those listed above. If you want to use other configurations, you have to change the configuration after running the deployment tool or modify the deployment tool.

This deployment tool has been tested under the following configurations.

* Nova-compute components are installed on multiple servers.

* OS: Ubuntu10.04, Ubuntu10.10 or Ubuntu 11.04

• Multiple network modes (VLAN Mode, Flat Mode)

Although we conducted extensive tests, we were unable to test every configuration. Please let us know any problems that occur in your environment by contacting us at https:// answers.launchpad.net/nova-deployment-tool. We will try to resolve any problem you send us and make the tool better for Stackers.

Note

The configurations, which are not described on this document, are Nova default settings. Note also that, although we have not done so ourselves, you should be able to change the network mode to flat DHCP mode and hypervisor to Xen if you follow the instructions in the Notes section below.

Overview of Deployment Tool Steps

You can install/test/uninstall Nova, Glance and Swift with the Nova deployment tool as follows, which is simply an overview. The detailed steps are in the sections that follow.

Deploy.py takes care of the details using puppet. Puppet is an automation tool with standardized scripts that manage a machine's configuration. See an Introduction to Puppet on the PuppetLabs site.

Install by typing the following command.

```
python deploy.py install
```

Confirm that the installation succeeded by typing the following command.

```
python deploy.py test
```

Uninstall by typing the following command.

```
python deploy.py uninstall
python deploy.py all = python deploy.py uninstall; python deploy.py install;
 python deploy.py test
```

Uninstall/install/test only Nova.

```
python deploy.py all nova
```

Uninstall/install/test only Swift.

```
python deploy.py all swift
```

Uninstall/install/test only Glance.

```
python deploy.py all glance
```

Installing the Deployment Tool

Type or copy/paste the following command to use the OpenStack PPA on all component servers.

```
sudo apt-get install python-software-properties -y
sudo add-apt-repository ppa:openstack-release/2011.2
sudo apt-get update
```

Set permissions to the deployment 'user'

Edit sudoers file to give the correct permissions to the 'user' running all the components. Type or copy/paste the visudo command to set 'user' (= nii in this document) as a sudouer on all nova component servers.

```
sudo visudo
```

Append the following lines to the visudo file, and then save the file.

```
nii       ALL=(ALL) NOPASSWD:ALL
nova      ALL=(ALL) NOPASSWD:ALL
```

Configure SSH

Next, we'll configure the system so that SSH works by generating public and private key pairs that provide credentials without a password intervention.

The Deployment tool needs to connect to all nova, glance and swift component servers without having the operator enter a password for any of the servers.

Type or copy/paste the following command to generate public and private key pairs on the server running the Nova deployment tool.

```
ssh-keygen -t rsa -N '' -f ~/.ssh/id_rsa
```

Copy this generated public key to all nova component servers.

Next, type or copy/paste the following commands to register the public keys on all nova component servers.

```
ssh-copy-id nii@(each nova component server name)
```

Download the code for the deployment tool next, and extract the contents of the compressed file.

```
wget http://launchpad.net/nova-deployment-tool/cactus/cactus1.3/+download/
nova-deployment-tool-cactus.tgz
tar xzvf nova-deployment-tool-cactus.tgz
```

Create Swift storage folder and mount device

First, create a Swift-storage folder and mount device on each swift-storage server.

The commands vary depending on which destination (Partition or Lookback device) is to be used.

The steps are detailed in the sections that follow.

"$storage_path" and "$storage_dev" are defined in "deploy.conf".

Partition

```
sudo apt-get install xfsprogs -y
sudo sh -c "echo '/dev/$storage_dev $storage_path/$storage_dev xfs
 noatime,nodiratime,nobarrier,logbufs=8 0 0' >> /etc/fstab"
sudo mount $storage_path/$storage_dev
```

Loopback device

```
sudo apt-get install xfsprogs -y
sudo mkdir -p $storage_path/$storage_dev
sudo dd if=/dev/zero of=/srv/swift-disk bs=1024 count=0 seek=1000000
sudo mkfs.xfs -i size=1024 /srv/swift-disk
sudo sh -c "echo '/srv/swift-disk $storage_path/$storage_dev xfs
 loop,noatime,nodiratime,nobarrier,logbufs=8 0 0' >> /etc/fstab"
sudo mount $storage_path/$storage_dev
```

Configuring the Deployment Tool

You must change the configuration file in order to execute the Nova deployment tool according to your environment and configuration design. In the unzipped files, edit conf/ deploy.conf to change the settings according to your environment and desired installation (single or multiple servers, for example).

Here are the definitions of the values which are used in deploy.conf.

default section

- puppet_server Name of server in which the puppet server is installed

- sh_user User name that is used to SSH into a nova component

nova section

- nova_api Name of server in which the nava-api component is installed

- nova_objectstore Name of server in which the nova-objectstore component is installed*

- nova_compute Name of server in which the nova-compute component is installed

- nova_scheduler Name of server in which the nova-scheduler component is installed

- nova_network Name of server in which the nova-network component is installed

- nova_volume Name of server in which the nova-volume component is installed

- euca2ools Name of server that runs the test sequence

- mysql Name of server in which mysql is installed

- glance_host Glance server name

- libvirt_type Virtualization type

- network_manager Network management class name

- image_service Image management class name

- network_interface Network interface that is used in the nova-compute component

- network_ip_range IP address range used by guest VMS. This value should be included in the values of fixed_range.

- volume_group LVM volume group name that is used in the nova volume component

- fixed_range Range of IP addresses used in all projects. If you want to change the value, please also change the IP addresses X.X.X.X of the command "nova-manage network create X.X.X.X ..." in file setup-network.sh, and the IP addresses should include the new value.

- network_size Number of IP addresses used by Guest VM in all projects

glance section

- glance Name of server in which the glance is installed

- default_store Image store that is used in glance. Available value: file, swift, s3

swift section

- swift_proxy Name of server in which the glance is installed

- swift_storage Name of server in which the swift=storage is installed

- account swift account name

- username swift user name

- password swift password

- storage_path Folder for saving account, container and object information in swift storage server

- storage_dev Device holding account, container and object information

- ring_builder_replicas Number of account, container, and object copies. The value has to be equal or less than the number of swift-storage servers.

- super_admin_key A key for creating swift users

If you install swift on Ubuntu 11.04, due to the bug https://bugs.launchpad.net/swift/+bug/796404 swift_proxy should be installed on the different machine from the machine where swift_storage will be installed.

Because of the current implementation architecture, you must load nova-api, nova-objectstore and euca2ools on a single server.

The following configuration information is an example. If you want to have multiple nova-computes, you can do so by nova_compute=ubuntu3, ubuntu8, for example. And if you

want to have multiple swift storage, you can do so by swift_storage=ubuntu3, ubuntu8, for example.

```
<begin ~/DeploymentTool/conf/deploy.conf>
[default]
puppet_server=ubuntu7
ssh_user=nii

[nova]
nova_api=ubuntu7
nova_objectstore=ubuntu7
nova_compute=ubuntu7
nova_scheduler=ubuntu7
nova_network=ubuntu7
nova_volume=ubuntu7
euca2ools=ubuntu7
mysql=ubuntu7

glance_host=ubuntu7

libvirt_type=kvm
network_manager=nova.network.manager.VlanManager
image_service=nova.image.glance.GlanceImageService

network_interface=eth0
network_ip_range=10.0.0.0/24

volume_group=ubuntu7
fixed_range=10.0.0.0/8
network_size=5000

[glance]
glance=ubuntu7
default_store=swift

[swift]
swift_proxy=ubuntu7
swift_storage=ubuntu7

account=system
username=root
password=testpass

storage_path=/srv/node
storage_dev=sdb1
ring_builder_replicas=1

super_admin_key=swauth
<end ~/DeploymentTool/conf/deploy.conf>
```

OpenStack Compute Installation Using VirtualBox, Vagrant, And Chef

Integration testing for distributed systems that have many dependencies can be a huge challenge. Ideally, you would have a cluster of machines that you could PXE boot to a base OS install and run a complete install of the system. Unfortunately not everyone has a bunch

of extra hardware sitting around. For those of us that are a bit on the frugal side, a whole lot of testing can be done with Virtual Machines. Read on for a simple guide to installing OpenStack Compute (Nova) with VirtualBox and Vagrant.

Installing VirtualBox

VirtualBox is virtualization software by Oracle. It runs on Mac/Linux/Windows and can be controlled from the command line. Note that we will be using VirtualBox 4.0 and the vagrant prerelease.

OSX

```
curl -O http://download.virtualbox.org/virtualbox/4.0.2/
VirtualBox-4.0.2-69518-OSX.dmg
open VirtualBox-4.0.2-69518-OSX.dmg
```

Ubuntu Maverick

```
wget -q http://download.virtualbox.org/virtualbox/debian/oracle_vbox.asc -O- |
 sudo apt-key add -
echo "deb http://download.virtualbox.org/virtualbox/debian maverick contrib" |
 sudo tee /etc/apt/sources.list.d/virtualbox.list
sudo apt-get update
sudo apt-get install -y virtualbox-4.0
```

Ubuntu Lucid

```
wget -q http://download.virtualbox.org/virtualbox/debian/oracle_vbox.asc -O- |
 sudo apt-key add -
echo "deb http://download.virtualbox.org/virtualbox/debian lucid contrib" |
 sudo tee /etc/apt/sources.list.d/virtualbox.list
sudo apt-get update
sudo apt-get install -y virtualbox-4.0
```

Install RubyGems

The following instructions for installing Vagrant use RubyGems for the installation commands. You can download RubyGems from http://rubygems.org/pages/download.

Get the Vagrant Pre-release

OSX

```
sudo gem update -- system
sudo gem install vagrant -- pre
```

Ubuntu Maverick

```
sudo gem install vagrant --pre
sudo ln -s /var/lib/gems/1.8/bin/vagrant /usr/local/bin/vagrant
```

Ubuntu Lucid

```
wget http://production.cf.rubygems.org/rubygems/rubygems-1.3.6.zip
sudo apt-get install -y unzip
unzip rubygems-1.3.6.zip
cd rubygems-1.3.6
```

```
sudo ruby setup.rb
sudo gem1.8 install vagrant --pre
```

Get the Chef Recipes

```
cd ~
git clone https://github.com/ansolabs/openstack-cookbooks/openstack-
cookbooks.git
```

Set Up Some Directories

```
mkdir aptcache
mkdir chef
cd chef
```

Get the chef-solo Vagrant file

Provisioning for vagrant can use chef-solo, chef-server, or puppet. We're going to use chef-solo for the installation of OpenStack Compute.

```
curl -o Vagrantfile https://raw.github.com/gist/786945/solo.rb
```

Running OpenStack Compute within a Vagrant Instance

Installing and running OpenStack Compute is as simple as typing "vagrant up"

```
vagrant up
```

In 3-10 minutes, your vagrant instance should be running. NOTE: Some people report an error from vagrant complaining about MAC addresses the first time they vagrant up. Doing `vagrant up` again seems to resolve the problem.

```
vagrant ssh
```

Now you can run an instance and connect to it:

```
. /vagrant/novarc
euca-add-keypair test > test.pem
chmod 600 test.pem
euca-run-instances -t m1.tiny -k test ami-tty
# wait for boot (euca-describe-instances should report running)
ssh -i test.pem root@10.0.0.3
```

Yo, dawg, your VMs have VMs! That is, you are now running an instance inside of OpenStack Compute, which itself is running inside a VirtualBox VM.

When the you are finished, you can destroy the entire system with vagrant destroy. You will also need to remove the .pem files and the novarc if you want to run the system again.

```
vagrant destroy
rm *.pem novarc
```

Using the dashboard

The OpenStack Dashboard should be running on 192.168.86.100. You can login using username: admin, password: vagrant.

8. 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, Nova 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, Nova supports three kinds of networks, implemented in three "Network Manager" types respectively: Flat Network Manager, Flat DHCP Network Manager, and 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.

Nova has a concept of Fixed IPs and Floating IPs. Fixed IPs are assigned to an instance on creation and stay the same until the instance is explicitly terminated. Floating IPs are IP addresses that can be dynamically associated with an instance. This 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 (named br100) 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, you start a DHCP server 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. In this mode, Compute does a bit more configuration in that it attempts to bridge into an ethernet device (eth0 by default). It will also run dnsmasq as a dhcpserver listening on this bridge. Instances receive their fixed IPs by doing a dhcpdiscover.

In both flat modes, the network nodes do not act as a default gateway. Instances are given public IP addresses. Compute nodes have iptables/ebtables entries created per project and instance to protect against IP/MAC address spoofing and ARP poisoning.

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.

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 nova-manage) 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 crl. This will keep revoked users from connecting and will disconnect any users that are connected with revoked certificates when their connection is renegotiated (every hour).

Creating a Cloudpipe Image

Making a cloudpipe image is relatively easy.

- # install openvpn on a base ubuntu image.

- # set up a server.conf.template in /etc/openvpn/

- # set up.sh in /etc/openvpn/

- # set down.sh in /etc/openvpn/

- # download and run the payload on boot from /etc/rc.local

- # setup /etc/network/interfaces

- # register the image and set the image id in your flagfile:

```
--vpn_image_id=ami-xxxxxxxx
```

- # you should set a few other flags to make vpns work properly:

```
--use_project_ca
--cnt_vpn_clients=5
```

When you use nova-manage to launch a cloudpipe for a user, it goes through the following process:

1. creates a keypair called <project_id>-vpn and saves it in the keys directory

2. creates a security group <project_id>-vpn and opens up 1194 and icmp

3. creates a cert and private key for the vpn instance and saves it in the CA/projects/ <project_id>/ directory

4. zips up the info and puts it b64 encoded as user data

5. launches an m1.tiny instance with the above settings using the flag-specified vpn image

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. (This will be turned into a nova-manage command or a flag soon.)

Certificates and Revocation

If the use_project_ca flag 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 euca-reboot-instances for example), but if you generate a new crl, you will have to terminate it and start it again using nova-manage vpn run. The cloudpipe instance always gets the first ip in the subnet and it can take up to 10 minutes 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. If you can't wait 10 minutes, you can manually update the ip with something like the following (use the right ip for the project):

```
            euca-terminate-instances <instance_id>
                mysql nova -e "update fixed_ips set allocated=0, leased=0,
 instance_id=NULL where fixed_ip='10.0.0.2'"
```

You also will need to terminate the dnsmasq running for the user (make sure you use the right pid file):

```
sudo kill `cat /var/lib/nova/br100.pid`
```

Now you should be able to re-run the vpn:

```
nova-manage vpn run <project_id>
```

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.

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 flag in nova.conf.

2. Use the `nova-manage network create networkname 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 flag

• –network_manager = nova.network.manager.VlanManager

   VLAN networking with DHCP. This is the Default if no network manager is defined in nova.conf.

Also, when you issue the nova-manage network create command, it uses the settings from the nova.conf flag file. Use the "nova-manage network create public 192.168.0.0/24 1 255" command to create the subnet that your VMs will run on. You specify public or private after the create command.

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 line:

–network_manager = nova.network.manager.FlatManager

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:

```
< begin /etc/network/interfaces >
# 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
< end /etc/network/interfaces >
```

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 8.1. Flat network, all-in-one server installation
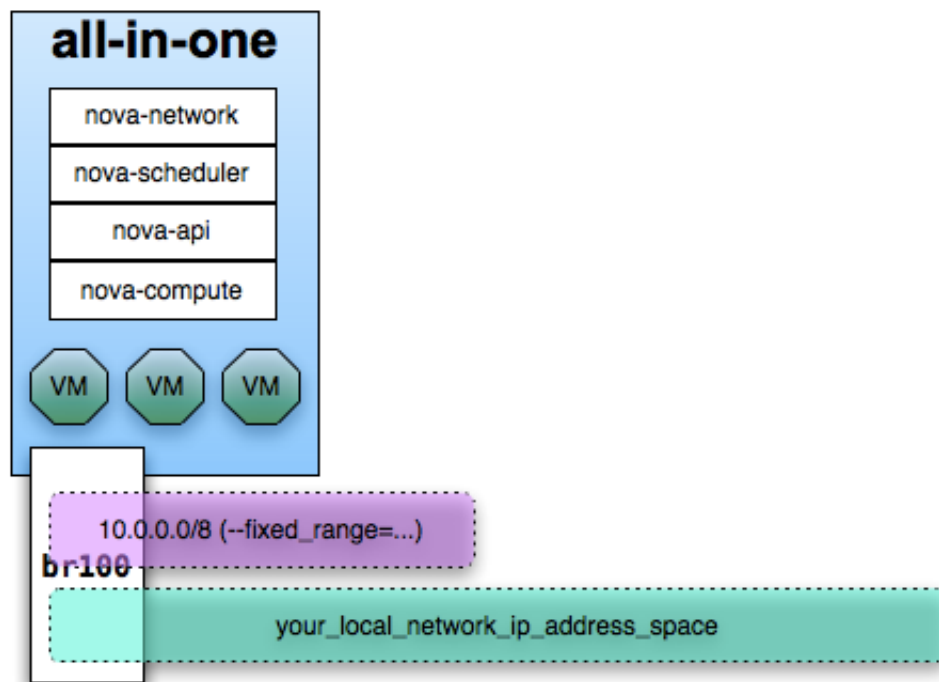
For multiple compute nodes with a single network adapter, which you can use for smoke testing or a proof of concept, this diagram represents the network setup.

Figure 8.2. Flat network, single interface, multiple servers
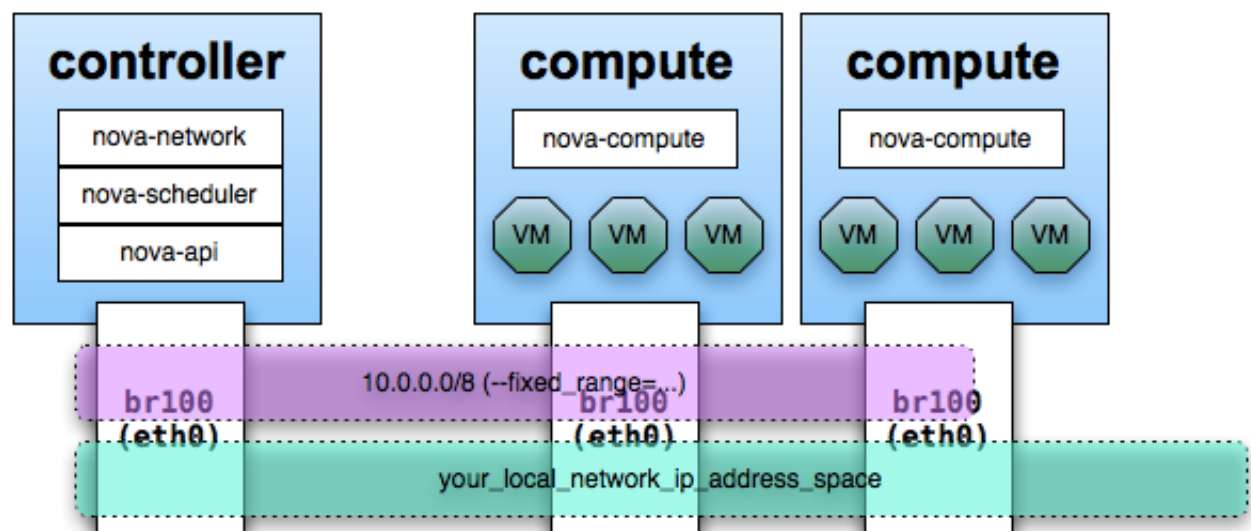
For multiple compute nodes with multiple network adapters, this diagram represents the network setup. You may want to use this setup for separate admin and data traffic.

Figure 8.3. Flat network, multiple interfaces, multiple servers
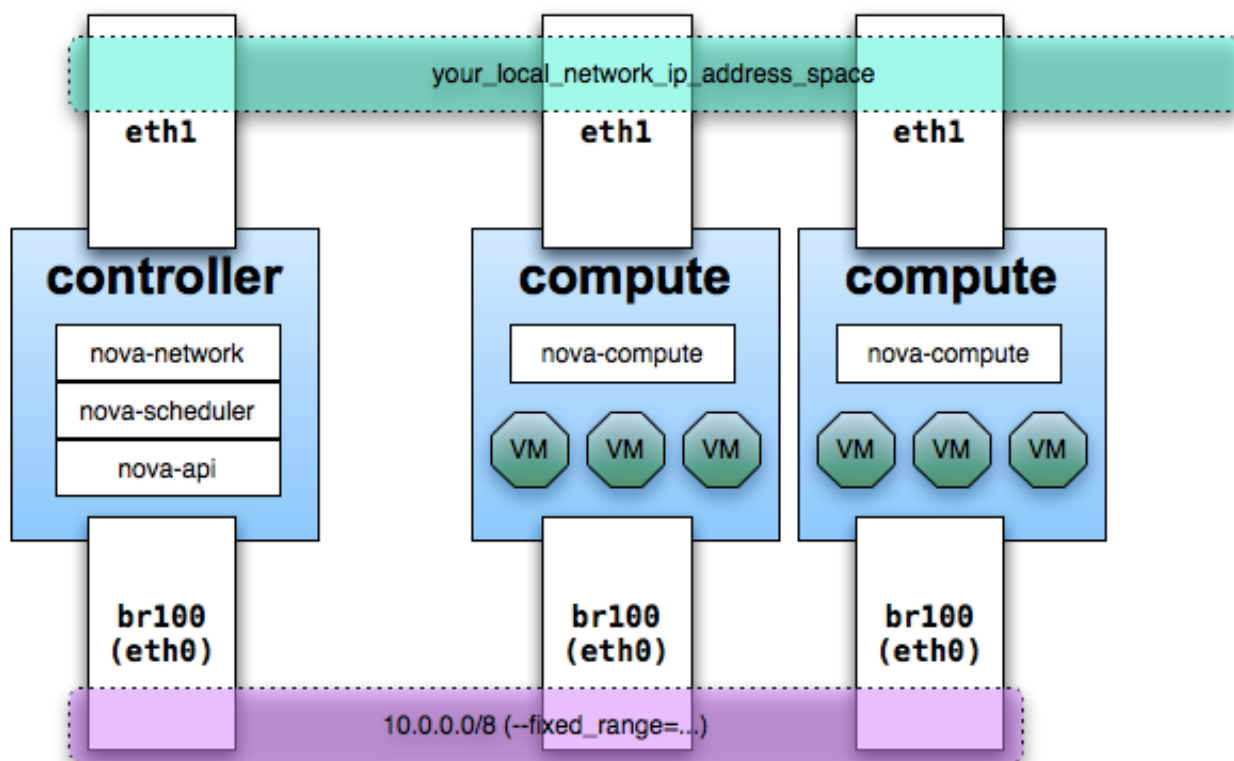
Configuring Flat DHCP Networking

With Flat DHCP, the host running nova-network acts as the gateway to the virtual nodes. You can run one nova-network per cluster. Set the flag –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 communicate with nova-network. You must also set the –flat_network_bridge setting to the name of the bridge (no default is set for it). The nova-network service will track leases and releases in the database so it knows if a VM instance has stopped properly configuring via DHCP. 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 do not need 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.

Visually, the setup looks like the diagram below:

Figure 8.4. Flat DHCP network, multiple interfaces, multiple servers

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. 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 flagfile. 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
–flat_network_dhcp_start=10.0.0.2
–flat_interface=eth2
–flat_injected=False
–public_interface=eth0
```

Integrate your network interfaces to match this configuration.

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 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 8.5. Single adaptor hosts, first route
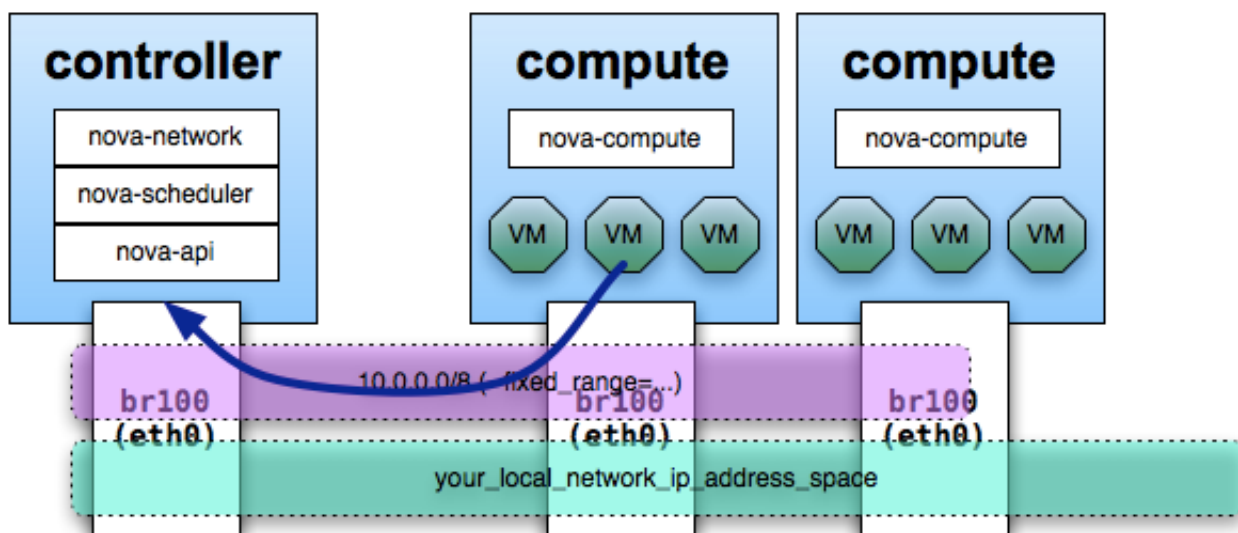
Next, the host on which nova-network is configured acts as a router and forwards the traffic out to the Internet.

Figure 8.6. Single adaptor hosts, second route
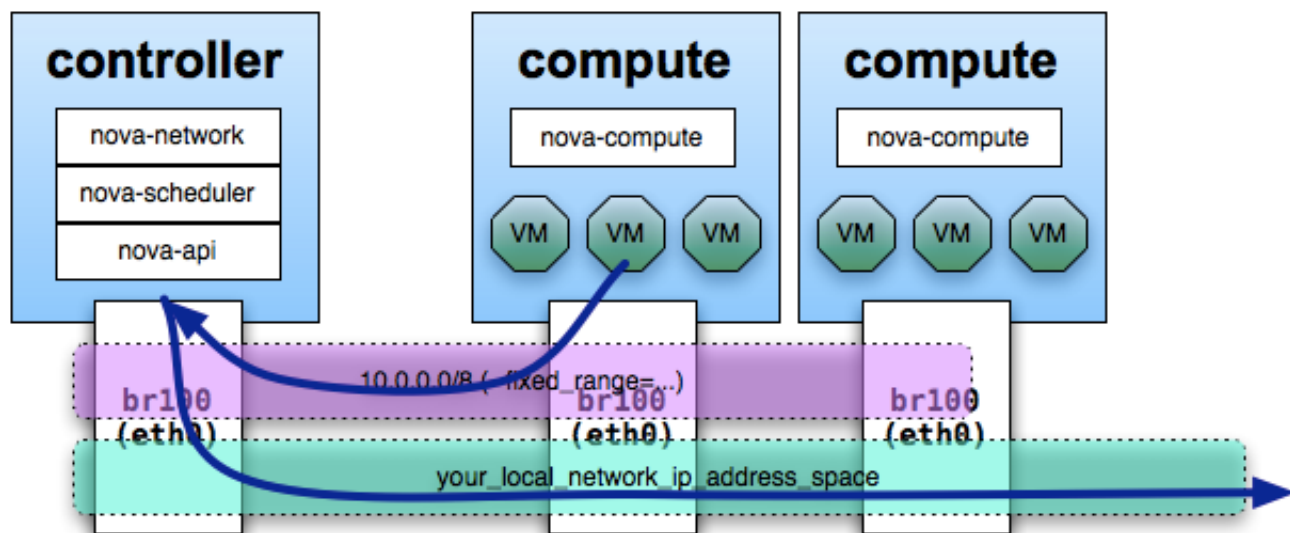
> ## Warning
>
> If you're using a single interface, then that interface (often eth0) needs to be set into promiscuous mode for the forwarding to happen correctly. This does not appear to be needed if you're running with physical hosts that have and use two interfaces.

Configuring VLAN Networking

In some networking environments, you may 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.

Using projects as a way to logically separate each VLAN, we can setup our cloud in this environment. Please note that you must have IP forwarding enabled for this network mode to work.

Obtain the parameters for each network. You may need to ask a network administrator for this information, including netmask, broadcast, gateway, ethernet device and VLAN ID.

You need to have networking hardware that supports VLAN tagging.

Please note that currently eth0 is hardcoded as the vlan_interface in the default flags. If you need to attach your bridges to a device other than eth0, you will need to add following flag to /etc/nova/nova.conf:

```
--vlan_interface=eth1
```

In VLAN mode, the setting for --network_size is the number of IPs per project as opposed to the FlatDHCP mode where --network_size indicates number of IPs in the entire fixed

range. For VLAN, the settings in nova.conf that affect networking are also –fixed_range, where the space is divided up into subnets of –network_size.

VLAN is the default networking mode for Compute, so if you have no –network_manager entry in your nova.conf file, you are set up for VLAN. To set your nova.conf file to VLAN, use this flag in /etc/nova/nova.conf:

```
–network_manager=nova.network.manager.VlanManager
```

For the purposes of this example walk-through, we will use the following settings. These are intentionally complex in an attempt to cover most situations:

• VLANs: 171, 172, 173 and 174

• IP Blocks: 10.1.171.0/24, 10.1.172.0/24, 10.1.173.0/24 and 10.1.174.0/24

• Each VLAN maps to its corresponding /24 (171 = 10.1.171.0/24, etc)

• Each VLAN will get its own bridge device, which is in the format br_$VLANID

• Each /24 has an upstream default gateway on .1

• The first 6 IPs in each /24 are reserved

First, create the networks that Compute can pull from using nova-manage commands:

```
nova-manage --flagfile=/etc/nova/nova.conf network create private
 10.1.171.0/24 1 256
nova-manage --flagfile=/etc/nova/nova.conf network create private
 10.1.172.0/24 1 256
nova-manage --flagfile=/etc/nova/nova.conf network create private
 10.1.173.0/24 1 256
nova-manage --flagfile=/etc/nova/nova.conf network create private
 10.1.174.0/24 1 256
```

Log in to the nova database to determine the network ID assigned to each VLAN:

```
select id,cidr from networks;
```

Update the DB to match your network settings. The following script will generate SQL based on the predetermined settings for this example. *You will need to modify this database update to fit your environment.*

```
if [ -z $1 ]; then
  echo "You need to specify the vlan to modify"
fi

if [ -z $2 ]; then
  echo "You need to specify a network id number (check the DB for the network
 you want to update)"
fi

VLAN=$1
ID=$2

cat > vlan.sql << __EOF_
update networks set vlan = '$VLAN' where id = $ID;
update networks set bridge = 'br_$VLAN' where id = $ID;
```

```
update networks set gateway = '10.1.$VLAN.7' where id = $ID;
update networks set dhcp_start = '10.1.$VLAN.8' where id = $ID;
update fixed_ips set reserved = 1 where address in ('10.1.$VLAN.1','10.1.
$VLAN.2','10.1.$VLAN.3','10.1.$VLAN.4','10.1.$VLAN.5','10.1.$VLAN.6','10.1.
$VLAN.7');
__EOF_
```

After verifying that the above SQL will work for your environment, run it against the nova database, once for every VLAN you have in the environment.

Next, create a project manager for the Compute project:

```
nova-manage --flagfile=/etc/nova/nova.conf user admin $username
```

Then create a project and assign that user as the admin user:

```
nova-manage --flagfile=/etc/nova/nova.conf project create $projectname
 $username
```

Finally, get the credentials for the user just created, which also assigns one of the networks to this project:)

```
nova-manage --flagfile=/etc/nova/nova.conf project zipfile $projectname
 $username
```

When you start nova-network, the bridge devices and associated VLAN tags will be created. When you create a new VM you must determine (either manually or programatically) which VLAN it should be a part of, and start the VM in the corresponding project.

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.

```
vconfig rem vlanNNN
ifconfig br_NNN down
brctl delbr br_NNN
```

Also, if users need to access the instances in their project across a VPN, a special VPN instance (code named cloudpipe) needs to be created. You can create the cloudpipe instance. The 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 should configure and run openvpn to run using the data from Compute.

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 flag in nova.conf for cloudpipes to work securely so that each project has its own Certificate Authority (CA).

Enabling Ping and SSH on VMs

Be sure you enable access to your VMs by using the 'euca-authorize' command. Below, you will find the commands to allow 'ping' and 'ssh' to your VMs:

```
euca-authorize -P icmp -t -1:-1 default
euca-authorize -P tcp -p 22 default
```

If you still cannot ping or SSH your instances after issuing the 'euca-authorize' 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: `killall dnsmasq; service nova-network restart`

Allocating and Associating IP Addresses with Instances

You can use Euca2ools commands to manage floating IP addresses used with Flat DHCP or VLAN networking.

To assign a reserved IP address to your project, removing it from the pool of available floating IP addresses, use `euca-allocate-address`. It'll return an IP address, assign it to the project you own, and remove it from the pool of available floating IP addresses.

To associate the floating IP to your instance, use `euca-associate-address -i [instance_id] [floating_ip]`.

When you want to return the floating IP to the pool, first use euca-disassociate-address [floating_ip] to disassociate the IP address from your instance, then use euca-deallocate-address [floating_ip] to return the IP to the pool of IPs for someone else to grab.

There are nova-manage commands that also help you manage the floating IPs.

nova-manage floating list - This command lists the floating IP addresses in the pool.

nova-manage floating create [hostname] [cidr] - This command creates specific floating IPs for a specific network host and either a single address or a subnet.

nova-manage floating destroy [hostname] [cidr] - This command removes floating IP addresses using the same parameters as the create command.

Associating a Public IP Address

OpenStack Compute uses NAT for public IPs. If you plan to use public IP addresses for your virtual instances, you must configure –public_interface=vlan100' in the nova.conf file so that Nova knows where to bind public IP addresses. Restart nova-network if you change nova.conf while the service is running. Also, ensure you have opened port 22 for the nova network.

You must add the IP address or block of public IP addresses to the floating IP list using the `nova-manage floating create` command. When you start a new virtual instance, associate one of the public addresses to the new instance using the euca-associate-address command.

These are the basic overall steps and checkpoints.

First, set up the public address.

```
nova-manage floating create my-hostname 68.99.26.170/31
euca-allocate-address 68.99.26.170
euca-associate-address -i i-1 68.99.26.170
```

Make sure the security groups are open.

```
root@my-hostname:~# euca-describe-groups
GROUP admin-project default default
PERMISSION admin-project default ALLOWS icmp -1 -1
FROM CIDR 0.0.0.0/0
PERMISSION admin-project default ALLOWS tcp 22 22
FROM CIDR 0.0.0.0/0
```

Ensure the NAT rules have been added to iptables.

```
-A nova-network-OUTPUT -d 68.99.26.170/32 -j DNAT --to-destination 10.0.0.3
-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.

```
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.

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
```

Existing High Availability Options for Networking

from Vish Ishaya

As you can see from the Flat DHCP diagram 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.

Option 1: 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.

Option 2: 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.

Option 3: HW Gateway

It is possible to tell dnsmasq to use an external gateway instead of acting as the gateway for the VMs. You can pass dhcpoption=3,<ip of gateway> to make the VMs use an external gateway. This will require some manual setup. The metadata IP forwarding rules will need to be set on the hardware gateway instead of the nova-network host. You will have to make sure to set up routes properly so that the subnet that you use for VMs is routable.

This offloads HA to standard switching hardware and it has some strong benefits. Unfortunately, nova-network is still responsible for floating IP natting and dhcp, so some failover strategy needs to be employed for those options.

New HA Option

Essentially, what the current options are lacking, is the ability to specify different gateways for different VMs. An agnostic approach to a better model might propose allowing multiple gateways per VM. Unfortunately this rapidly leads to some serious networking complications, especially when it comes to the natting for floating IPs. With a few assumptions about the problem domain, we can come up with a much simpler solution that is just as effective.

The key realization is that there is no need to isolate the failure domain away from the host where the VM is running. If the host itself goes down, losing networking to the VM is a non-issue. The VM is already gone. So the simple solution involves allowing each compute host to do all of the networking jobs for its own VMs. This means each compute host does NAT, dhcp, and acts as a gateway for all of its own VMs. While we still have a single point of failure in this scenario, it is the same point of failure that applies to all virtualized systems, and so it is about the best we can do.

So the next question is: how do we modify the Nova code to provide this option. One possibility would be to add code to the compute worker to do complicated networking setup. This turns out to be a bit painful, and leads to a lot of duplicated code between compute and network. Another option is to modify nova-network slightly so that it can run successfully on every compute node and change the message passing logic to pass the network commands to a local network worker.

Surprisingly, the code is relatively simple. A couple fields needed to be added to the database in order to support these new types of "multihost" networks without breaking the functionality of the existing system. All-in-all it is a pretty small set of changes for a lot of added functionality: about 250 lines, including quite a bit of cleanup. You can see the branch here: https://code.launchpad.net/~vishvananda/nova/ha-net/+merge/67078

The drawbacks here are relatively minor. It 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 3 above 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 8.7. High Availability Networking Option

In contrast with the earlier diagram, all the hosts in the system are running both nova-compute and nova-network. 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 requirements for configuring are the following: –multi_host flag 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.

Future of Networking

With the existing multi-nic code and the HA networking code, we have a pretty robust system with a lot of deployment options. This should be enough to provide deployers enough room to solve todays networking problems. Ultimately, we want to provide users the ability to create arbitrary networks and have real and virtual network appliances managed automatically. The efforts underway in the Quantum and Melange projects will help us reach this lofty goal, but with the current additions we should have enough flexibility to get us by until those projects can take over.

9. 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 flags 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 flag to FlatManager, FlatDHCPManager, or VlanManager (default is VLAN if no other is specified):

```
nova-network --network_manager=nova.network.manager.FlatManager
```

Starting Images

Once you have an installation, you want to get images that you can use in your Compute cloud. We've created a basic Ubuntu image for testing your installation. First you'll download the image, then use uec-publish-tarball to publish it:

```
image="ubuntu1010-UEC-localuser-image.tar.gz"
wget http://c0179148.cdn1.cloudfiles.rackspacecloud.com/ubuntu1010-UEC-
localuser-image.tar.gz
uec-publish-tarball $image [bucket-name] [hardware-arch]
```

Here's an example of what this command looks like with data:

```
 uec-publish-tarball ubuntu1010-UEC-localuser-image.tar.gz dub-bucket x86_64
```

The command in return should output three references: emi, eri and eki. You need to use the emi value (for example, "ami-zqkyh9th") for the euca-run-instances command.

Now you can schedule, launch and connect to the instance, which you do with tools from the Euca2ools on the command line. Create the emi value from the uec-publish-tarball command, and then you can use the euca-run-instances command.

One thing to note here, once you publish the tarball, it has to untar before you can launch an image from it. Using the 'euca-describe-images' command, wait until the state turns to "available" from "untarring.":

```
euca-describe-images
```

Depending on the image that you're using, you need a public key to connect to it. Some images have built-in accounts already created. Images can be shared by many users, so it is dangerous 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 root's authorized_keys file. 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.

Keypairs are created through the api and you use them as a parameter when launching an instance. They can be created on the command line using the euca2ools script euca-add-keypair. Refer to the man page for the available options. Example usage:

```
euca-add-keypair test > test.pem
chmod 600 test.pem
```

Now, you can run the instances:

```
euca-run-instances -k test -t m1.tiny ami-zqkyh9th
```

Here's a description of the parameters used above:

-t what type of image to create

-k name of the key to inject in to the image at launch

Optionally, you can use the -n parameter to indicate how many images of this type to launch.

The instance will go from "launching" to "running" in a short time, and you should be able to connect via SSH using the 'ubuntu' account, with the password 'ubuntu': (replace $ipaddress with the one you got from euca-describe-instances):

```
ssh ubuntu@$ipaddress
```

The 'ubuntu' user is part of the sudoers group, so you can escalate to 'root' via the following command:

```
sudo -i
```

Deleting Instances

When you are done playing with an instance, you can tear the instance down using the following command (replace $instanceid with the instance IDs from above or look it up with euca-describe-instances):

```
euca-terminate-instances $instanceid
```

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=on -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

Since, Fedora does not ship with cloud-init or an equivalent, you will need to take a few steps to have the instance fetch the meta data like ssh keys etc.

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 "***********************"
```

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 Imaging Server glance. 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.

uec-publish-image, like several other commands from euca2ools, 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 'euca-describe-images' 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.

```
euca-bundle-image -i server.img
euca-upload-bundle -b mybucket -m /tmp/server.img.manifest.xml
euca-register mybucket/server.img.manifest.xml
```

Image Listing

The status of the images that have been uploaded can be viewed by using euca-describe-images command. The output should like this:

```
localadmin@client1:~$ euca-describe-images

IMAGE   ari-7bfac859    bucket1/initrd.img-2.6.38-7-server.manifest.xml   css
   available          private        x86_64        ramdisk

IMAGE   ami-5e17eb9d    bucket1/serverfinal.img.manifest.xml       css
 available            private        x86_64   machine     aki-3d0aeb08
 ari-7bfac859

IMAGE   aki-3d0aeb08    bucket1/vmlinuz-2.6.38-7-server.manifest.xml     css
   available          private        x86_64        kernel

localadmin@client1:~$
```

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 1024 -cdrom win2k8_dvd.iso -drive
 file=windowsserver.img,if=virtio,boot=on -fda virtio-win-1.1.16.vfd -boot d -
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 euca-authorize command to open up port 3389 as described in the chapter on "Security".

Shut-down the VM and upload the image to OpenStack

```
euca-bundle-image -i windowsserver.img
euca-upload-bundle -b mybucket -m /tmp/windowsserver.img.manifest.xml
euca-register mybucket/windowsserver.img.manifest.xml
```

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 the Cloud

There are two main tools that a system administrator will find useful to manage their cloud; the nova-manage command or the Euca2ools command line commands.

With the Diablo release, the nova-manage command has been deprecated and you must specify if you want to use it by using the –use_deprecated_auth flag in nova.conf. You must also use the modified middleware stack that is commented out in the default paste.ini file.

The nova-manage command may only be run by users with admin privileges. Commands for euca2ools can be used by all users, though specific commands may be restricted by Role Based Access Control in the deprecated nova auth system.

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 user creation, vpn management, and much more.

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

Command categories are: account, agent, config, db, fixed, flavor, floating, host, instance_type, image, network, project, role, service, shell, user, version, vm, volume, and vpn.

You can also run with a category argument such as user to see a list of all commands in that category: nova-manage user

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. This can be easily accomplished using the user create or user admin commands in nova-manage. user create will create a regular user, whereas user admin will create an admin user. The syntax of the command is nova-manage user create username [access] [secretword]. For example:

```
nova-manage user create john my-access-key a-super-secret-key
```

If you do not specify an access or secret key, a random uuid will be created automatically.

Credentials

Nova can generate a handy set of credentials for a user. These credentials include a CA for bundling images and a file for setting environment variables to be used by euca2ools. If you don't need to bundle images, just the environment script is required. You can export one with the project environment command. The syntax of the command is nova-manage project environment project_id user_id [filename]. If you don't specify a filename, it will be exported as novarc. After generating the file, you can simply source it in bash to add the variables to your environment:

```
    nova-manage project environment john_project john
  . novarc
```

If you do need to bundle images, you will need to get all of the credentials using project zipfile. Note that zipfile will give you an error message if networks haven't been created

yet. Otherwise zipfile has the same syntax as environment, only the default file name is nova.zip. Example usage:

```
nova-manage project zipfile john_project john
unzip nova.zip
. novarc
```

Role Based Access Control

Roles control the API actions that a user is allowed to perform. For example, a user cannot allocate a public ip without the netadmin role. It is important to remember that a users de facto permissions in a project is the intersection of user (global) roles and project (local) roles. So for john to have netadmin permissions in his project, he needs to separate roles specified. You can add roles with role add. The syntax is nova-manage role add user_id role [project_id]. Let's give john the netadmin role for his project:

```
nova-manage role add john netadmin
nova-manage role add john netadmin john_project
```

Role-based access control (RBAC) is an approach to restricting system access to authorized users based on an individual's role within an organization. Various employee functions require certain levels of system access in order to be successful. These functions are mapped to defined roles and individuals are categorized accordingly. Since users are not assigned permissions directly, but only acquire them through their role (or roles), management of individual user rights becomes a matter of assigning appropriate roles to the user. This simplifies common operations, such as adding a user, or changing a user's department.

Nova's rights management system employs the RBAC model and currently supports the following five roles:

• Cloud Administrator. (cloudadmin) Users of this class enjoy complete system access.

• IT Security. (itsec) This role is limited to IT security personnel. It permits role holders to quarantine instances.

• System Administrator. (sysadmin)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) Users with this role are permitted to allocate and assign publicly accessible IP addresses as well as create and modify firewall rules.

• Developer. This is a general purpose role that is assigned to users by default.

• Project Manager. (projectmanager) This is a role that is assigned upon project creation and can't be added or removed, but this role can do anything a sysadmin can do.

RBAC management is exposed through the dashboard for simplified user management.

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 that

Amazon EC2 offers, 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 present 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 (we use here the euca2ools, but the same explanation goes with the API):

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

2. The volume is attached to an instance via $euca-attach-volume; 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-compute, 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 the way nova-volume works, but it is essential for nova-volumes to work that the mode you are currently using is set up. Please refer to the Section 7 "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 three 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-volumes installation.

A- Install nova-volumes 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-volumes**

  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', then select the new partition you made.

  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

Again :
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-volumes on the compute nodes

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

```
euca-create-volume
```

```
euca-attach-volume
```

```
euca-detach-volume
```

```
euca-delete-volume
```

- **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 your compute-nodes only :**

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

You have to enable it so the startut script (/etc/init.d/open-iscsi) will work :

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

Then run :

```
service iscsitarget start
```

- **Configure nova.conf flag file**

Edit your nova.conf to include a new flag, "–iscsi_ip_prefix=192.168." The flag will be used by the compute node when the iSCSI discovery will be performed and the session created. The prefix based on the two first bytes will allows the iSCSI discovery to use all the available routes (also known as multipathing) to the iSCSI server (eg. nova-volumes) into your network. We will see into the "Troubleshooting" section how to deal with ISCSI sessions.

- **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. You will be able to use the euca2ools related to volumes interactions (see above).

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:

```
euca-create-volume -s 7 -z nova  (-s refers to the size of the volume in GB,
 and -z is the default zone (usually nova))
```

You should get some output similar to this:

```
VOLUME vol-0000000b 7 creating (wayne, None, None, None) 2011-02-11
 06:58:46.941818
```

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

```
euca-attach-volume -i i-00000008 -d /dev/vdb vol-00000009
```

(-i refers to the instance you will attach the volume to, -d is the mountpoint **(on the compute-node !** and then the volume name.)

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 :

```
iscsciadm -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-0000000b
```

If you do not get any errors, it is time to login to instance 'i-00000008' and see if the new space is there. 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' from i-00000008:

```
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 denote it as 1st disk partition.

4. Press ENTER twice to accept the default of 1st 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 'euca' 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-volumes 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 "15- already exists"*

```
"ProcessExecutionError: Unexpected error while running command.\nCommand:
 sudo iscsiadm -m node -T iqn.2010-10.org.openstack:volume-00000001 -p
10.192.12.34:3260 --login\nExit code: 255\nStdout: 'Logging in to [iface:
 default, target: iqn.2010-10.org.openstack:volume-00000001, portal:
10.192.12.34,3260]\\n'\nStderr: 'iscsiadm: Could not login to [iface:
 default, target: iqn.2010-10.org.openstack:volume-00000001,
portal:10.192.12.34,3260]: openiscsiadm: initiator reported error (15 -
 already exists)\\n'\n"]
```

This errors happens sometimes when you run an euca-detach-volume and euca-attach-volume and/ or try to attach another volume to an instance. It happens when the

compute node has a running session while you try to attach a volume by using the same IQN. You could check that by running :

```
iscsiadm -m session
```

You should have a session with the same name that the compute is trying to open. Actually, it seems to be related to the several routes available for the iSCSI exposition, those routes could be seen by running on the compute node :

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

You should see for a volume multiple addresses to reach it. The only known workaround to that is to change the "–iscsi_ip_prefix" flag and use the 4 bytes (full IP) of the nova-volumes server, eg :

```
"–iscsi_ip_prefix=192.168.2.1
```

You'll have then to restart both nova-compute and nova-volume services.

• *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 will likely gives you 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-0000000e
tcp: [2] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-00000010
tcp: [3] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-00000011
tcp: [4] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-0000000a
tcp: [5] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-00000012
tcp: [6] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-00000007
tcp: [7] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-00000009
tcp: [9] 172.16.40.244:3260,1 iqn.2010-10.org.openstack:volume-00000014
```

I would close the session number 9 if I want to free the volume 00000014.

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

```
VOLUME  vol-00000014    30    nova    in-use (nuage-and-co, nova-cc1,
 i-0000009a[nova-cn1], \/dev\/sdb)   2011-07-18T12:45:39Z
```

You now have to inform it 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
```

Then, we get some information from the table "volumes" :

```
 mysql> select id,created_at, size, instance_id, status, attach_status,
 display_name from volumes;
 +----+-------------------+------+------------+---------------
 +---------------+--------------+
 | id | created_at        | size | instance_id | status        |
  attach_status | display_name |
 +----+-------------------+------+------------+---------------
 +---------------+--------------+
 |  1 | 2011-06-08 09:02:49 |    5 |          0 | available     | detached
        | volume1       |
 |  2 | 2011-06-08 14:04:36 |    5 |          0 | available     | detached
        | NULL          |
 |  3 | 2011-06-08 14:44:55 |    5 |          0 | available     | detached
        | NULL          |
```

```
|  4 | 2011-06-09 09:09:15 |    5 |        0 | error_deleting | detached
     | NULL          |
|  5 | 2011-06-10 08:46:33 |    6 |        0 | available      | detached
     | NULL          |
|  6 | 2011-06-10 09:16:18 |    6 |        0 | available      | detached
     | NULL          |
|  7 | 2011-06-16 07:45:57 |   10 |      157 | in-use         | attached
     | NULL          |
|  8 | 2011-06-20 07:51:19 |   10 |        0 | available      | detached
     | NULL          |
|  9 | 2011-06-21 08:21:38 |   10 |      152 | in-use         | attached
     | NULL          |
| 10 | 2011-06-22 09:47:42 |   50 |      136 | in-use         | attached
     | NULL          |
| 11 | 2011-06-30 07:30:48 |   50 |        0 | available      | detached
     | NULL          |
| 12 | 2011-06-30 11:56:32 |   50 |        0 | available      | detached
     | NULL          |
| 13 | 2011-06-30 12:12:08 |   50 |        0 | error_deleting | detached
     | NULL          |
| 14 | 2011-07-04 12:33:50 |   30 |      155 | in-use         | attached
     | NULL          |
| 15 | 2011-07-06 15:15:11 |    5 |        0 | error_deleting | detached
     | NULL          |
| 16 | 2011-07-07 08:05:44 |   20 |      149 | in-use         | attached
     | NULL          |
| 20 | 2011-08-30 13:28:24 |   20 |      158 | in-use         | attached
     | NULL          |
| 17 | 2011-07-13 19:41:13 |   20 |      149 | in-use         | attached
     | NULL          |
| 18 | 2011-07-18 12:45:39 |   30 |      154 | in-use         | attached
     | NULL          |
| 19 | 2011-08-22 13:11:06 |   50 |        0 | available      | detached
     | NULL          |
| 21 | 2011-08-30 15:39:16 |    5 |     NULL | error_deleting | detached
     | NULL          |
+----+---------------------+------+------------+----------------
+--------------+-------------+
21 rows in set (0.00 sec)
```

Once you get the volume id, you will have to run the following sql queries (let's say, my volume 14 as the id number 21 :

```
        mysql> update volumes set mountpoint=NULL where id=21;
        mysql> update volumes set status="available" where status
"error_deleting" where id=21;
        mysql> update volumes set attach_status="detached" where id=21;
        mysql> update volumes set instance_id=0 where id=21;
```

Now if you run again `euca-describe-volumes`from the cloud controller, you should see an available volume now :

```
VOLUME   vol-00000014      30      nova     available (nuage-and-co, nova-cc1,
 None, None)  2011-07-18T12:45:39Z
```

You can now proceed to the volume attachment again!

Using Live Migration

Before starting live migration, check "Configuring Live Migration" sections.

Live migration provides a scheme to migrate running instances from one OpenStack Compute server to another OpenStack Compute server. No visible downtime and no transaction loss is the ideal goal. This feature can be used as depicted below.

- First, make sure any instances running on a specific server.

```
# euca-describe-instances
Reservation:r-2raqmabo
RESERVATION    r-2raqmabo     admin    default
INSTANCE       i-00000003     ami-ubuntu-lucid      a.b.c.d  e.f.g.h
 running      testkey (admin, HostB)  0                m1.small     2011-02-15
 07:28:32    nova
```

In this example, i-00000003 is running on HostB.

- Second, pick up other server where instances are migrated 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 onto it.

- Third, check HostC has enough resource for live migration.

```
# nova-manage service update_resource HostC
# nova-manage service describe_resource HostC
HOST            PROJECT     cpu    mem(mb) disk(gb)
HostC(total)                16     32232     878
HostC(used)                 13     21284     442
HostC           p1           5     10240     150
HostC           p2           5     10240     150
.....
```

Remember to use update_resource first, then describe_resource. Otherwise, Host(used) is not updated.

- **cpu:**the nuber of cpu

- **mem(mb):**total amount of memory (MB)

- **disk(gb)**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 and under** is used resource per project.

- Finally, live migration

```
# nova-manage vm live_migration i-00000003 HostC
Migration of i-00000001 initiated. Check its progress using euca-describe-
instances.
```

Make sure instances are migrated successfully with euca-describe-instances. If instances are still running on HostB, check logfiles(src/dest nova-compute and nova-scheduler)

Reference for Flags in nova.conf

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

Table 9.1. Description of common nova.conf flags (nova-api, nova-compute)

Flag	Default	Description
–ajax_console_proxy_port	default: '8000'	Port value; port to which the ajax console proxy server binds
–ajax_console_proxy_topic	default: 'ajax_proxy'	String value; Topic that the ajax proxy nodes listen on
–ajax_console_proxy_url	default: 'http://127.0.0.1:8000'	IP address plus port value; Location of the ajax console proxy and port
–auth_token_ttl	default: '3600'	Seconds; Amount of time for auth tokens to linger, must be an integer value
–aws_access_key_id	default: 'admin'	Username; ID that accesses AWS if necessary
–aws_secret_access_key	default: 'admin'	Password key; The secret access key that pairs with the AWS ID for connecting to AWS if necessary
–compute_manager	default: 'nova.compute.manager.ComputeManager'	String value; Manager for Compute which handles remote procedure calls relating to creating instances
–compute_topic	default: 'compute'	String value; Names the topic that compute nodes listen on
–connection_type	default: 'libvirt'	String value libvirt, xenapi or fake; Virtualization driver for spawning instances
–console_manager	default: 'nova.console.manager.ConsoleProxyManager'	String value; Manager for console proxy
–console_topic	default: 'console'	String value; The topic console proxy nodes listen on
–control_exchange	default:nova	String value; Name of the main exchange to connect to
–default_image	default: 'ami-11111'	Name of an image; Names the default image to use, testing purposes only
–db_backend	default: 'sqlalchemy'	The backend selected for the database connection

Flag	Default	Description
--db_driver	default: 'nova.db.api'	The drive to use for database access
--default_instance_type	default: 'm1.small'	Name of an image; Names the default instance type to use, testing purposes only
--default_log_levels	default: 'amqplib=WARN,sqlalchemy=WARN,eventlet.wsgi.server=WARN'; List of logger=LEVEL pairs	Pair of named loggers and level of messages to be logged; List of logger=LEVEL pairs
--default_project	default: 'openstack'	Name of a project; Names the default project for openstack
--ec2_dmz_host	default: '$my_ip'	IP Address; Internal IP of API server (a DMZ is shorthand for a demilitarized zone)
--ec2_host	default: '$my_ip'	IP Address; External-facing IP of API server
--ec2_listen_port	default: '8773'	Port value; Port that the server is listening on so you can specify a listen_host / port value for the server (not for clients).
--ec2_path	default: '/services/Cloud'	String value; Suffix for EC2-style URL where nova-api resides
--ec2_port	default: '8773'	Port value; Cloud controller port (where nova-api resides)
--ec2_scheme	default: 'http'	Protocol; Prefix for EC2-style URLs where nova-api resides
--ec2_url	none	Deprecated - HTTP URL; Location to interface nova-api. Example: http://184.106.239.134:8773/services/Cloud
--flat_injected	default: 'false'	Indicates whether Compute (Nova) should use attempt to inject IPv6 network configuration information into the guest. It attempts to modify /etc/network/interfaces and currently only works on Debian-based systems.
--fixed_ip_disassociate_timeout	default: '600'	Integer: Number of seconds after which a deallocated ip is disassociated.
--fixed_range	default: '10.0.0.0/8'	Fixed IP address block of addresses from which a set of iptables rules is created
--fixed_range_v6	default: 'fd00::/48'	Fixed IPv6 address block of addresses
--[no]flat_injected	default: 'true'	Indicates whether to attempt to inject network setup into guest; network injection only works for Debian systems
--flat_interface	default: ''	FlatDhcp will bridge into this interface
--flat_network_bridge	default: ''	Bridge for simple network instances, formerly defaulted to br100; required setting for Flat DHCP
--flat_network_dhcp_start	default: '10.0.0.2'	Starting IP address for the DHCP server to start handing out IP addresses when using FlatDhcp
--flat_network_dns	default: '8.8.4.4'	DNS for simple network
--floating_range	default: '4.4.4.0/24'	Floating IP address block

Flag	Default	Description
–[no]fake_network	default: 'false'	Indicates whether Compute (Nova) should use fake network devices and addresses
–[no]enable_new_services	default: 'true'	Services to be added to the available pool when creating services using nova-manage
–[no]fake_rabbit	default: 'false'	Indicates whether Compute (Nova) should use a fake rabbit server
–glance_api_servers	default: '$my_ip:9292'	List of Glance API hosts. Each item may contain a host (or IP address) and port of an OpenStack Compute Image Service server (project's name is Glance)
-?, –[no]help		Show this help.
–[no]helpshort		Show usage only for this module.
–[no]helpxml		Show this help, but with XML output instead of text
–host	default: ''	String value; Name of the node where the cloud controller is hosted

–image_service	default: 'nova.image.s3.S3ImageService'	The service to use for retrieving and searching for images. Images must be registered using euca2ools. Options:
		
		• nova.image.s3.S3ImageService
		
		  S3 backend for the Image Service.
		
		• nova.image.local.LocalImageService
		
		  Image service storing images to local disk. It assumes that image_ids are integers. This is the default setting if no image manager is defined here.
		
		• nova.image.glance.GlanceImageService
		
		  Glance back end for storing and retrieving images; See http://glance.openstack.org for more info.

–image_decryption_dir	default: 'tmp/'	Parent directory for the temporary directory used for image decryption. Ensure the user has correct permissions to access this directory when decrypting images.
–instance_name_template	default: 'instance-%08x'	Template string to be used to generate instance names.
–libvirt_type	default: kvm	String: Name of connection to a hypervisor through libvirt. Supported options are kvm, qemu, uml, and xen.
–lock_path	default: none	Directory path: Writeable path to store lock files.
–logfile	default: none	Output to named file.
–logging_context_format_string	default: '%(asctime)s %(levelname)s %(name)s [%(request_id)s %(user)s %(project)s] %(message)s'	The format string to use for log messages with additional context.
–logging_debug_format_suffix	default: 'from %(processName)s (pid=%(process)d) %(funcName)s %(pathname)s:%(lineno)d'	The data to append to the log format when level is DEBUG.

Flag	Default	Description
–logging_default_format_string	default: '%(asctime)s %(levelname)s %(name)s [-] %(message)s'	The format string to use for log messages without context.
–logging_exception_prefix	default: '(%(name)s): TRACE: '	String value; Prefix each line of exception output with this format.
–my_ip	default: ''	IP address; Cloud controller host IP address.
–network_manager	default: 'nova.network.manager.VlanManager'	Configures how your controller will communicate with additional OpenStack Compute nodes and virtual machines. Options: • nova.network.manager.FlatManager Simple, non-VLAN networking • nova.network.manager.FlatDHCPManager Flat networking with DHCP • nova.network.manager.VlanManager VLAN networking with DHCP; This is the Default if no network manager is defined here in nova.conf.
–network_driver	default: 'nova.network.linux_net'	String value; Driver to use for network creation.
–network_host	default: 'preciousroy.hsd1.ca.comcast.net'	String value; Network host to use for ip allocation in flat modes.
–network_size	default: '256'	Integer value; Number of addresses in each private subnet.
–num_networks	default: '1000'	Integer value; Number of networks to support.
–network_topic	default: 'network'	String value; The topic network nodes listen on.
–node_availability_zone	default: 'nova'	String value; Availability zone of this node.
–null_kernel	default: 'nokernel'	String value; Kernel image that indicates not to use a kernel, but to use a raw disk image instead.
–osapi_host	default: '$my_ip'	IP address; IP address of the API server.
–osapi_listen_port	default: '8774'	Port value; Port for the OpenStack Compute API to listen on.
–osapi_path	default: '/v1.0/'	
–osapi_port	default: '8774'	Integer value; Port open for the OpenStack API server.
–osapi_scheme	default: 'http'	Protocol; Prefix for the OpenStack API URL.
–periodic_interval	default: '60'	Integer value; Seconds between running periodic tasks.
–pidfile	default: ''	String value; Name of pid file to use for this service (such as the nova-compute service).
–rabbit_host	default: 'localhost'	IP address; Location of rabbitmq installation.
–rabbit_max_retries	default: '12'	Integer value; Rabbit connection attempts.

Flag	Default	Description
–rabbit_password	default: 'guest'	String value; Password for the Rabbitmq server.
–rabbit_port	default: '5672'	Integer value; Port where Rabbitmq server is running/listening.
–rabbit-retry-interval	default: '10'	Integer value: Rabbit connection retry interval.
–rabbit_userid	default: 'guest'	String value; User ID used for Rabbit connections.
–region_list	default: ''	Comma-delimited pairs; List of region = fully qualified domain name pairs separated by commas.
–report_interval	default: '10'	Integer value; Seconds between nodes reporting state to the data store.
–routing_source_ip	default: '10'	IP address; Public IP of network host. When instances without a floating IP hit the Internet, traffic is snatted to this IP address.
–s3_dmz	default: '$my_ip'	IP address; For instances internal IP (a DMZ is shorthand for a demilitarized zone)
–s3_host	default: '$my_ip'	IP address: IP address of the S3 host for infrastructure. Location where OpenStack Compute is hosting the objectstore service, which will contain the virtual machine images and buckets.
–s3_port	default: '3333'	Integer value; Port where S3 host is running
–scheduler_manager	default: 'nova.scheduler.manager.SchedulerManager'	Manager for the scheduler for Compute (Nova)
–scheduler_topic	default: 'scheduler'	String value; The topic scheduler nodes listen on.
–sql_connection	default: 'sqlite:///$state_path/ nova.sqlite'	IP address; Location of OpenStack Compute SQL database
–sql_idle_timeout	default: '3600'	
–sql_max_retries	default: '12'	Integer value; Number of attempts on the SQL connection
–sql_retry_interval	default: '10'	Integer value; Retry interval for SQL connections
–state_path	default: '/usr/lib/pymodules/ python2.6/nova/../'	Top-level directory for maintaining Nova's state
–use_deprecated_auth	default: 'false'	Set to 1 or true to turn on; Determines whether to use the deprecated nova auth system or Keystone as the auth system
–use_ipv6	default: 'false'	Set to 1 or true to turn on; Determines whether to use IPv6 network addresses
–use_s3	default: 'true'	Set to 1 or true to turn on; Determines whether to get images from s3 or use a local copy
–verbose	default: 'false'	Set to 1 or true to turn on; Optional but helpful during initial setup
–vlan_interface	default: 'eth0'	This is the interface that VlanManager uses to bind bridges and vlans to.

Flag	Default	Description
–vlan_start	default: '100'	Integer; First VLAN for private networks.
–vpn_image_id	default: 'ami-cloudpipe'	AMI (Amazon Machine Image) for cloudpipe VPN server
–vpn_key_suffix	default: '-vpn'	This is the interface that VlanManager uses to bind bridges and VLANs to.

Table 9.2. Description of nova.conf flags specific to nova-volume

Flag	Default	Description
–iscsi_ip_prefix	default: ''	IP address or partial IP address; Value that differentiates the IP addresses using simple string matching, so if all of your hosts are on the 192.168.1.0/24 network you could use –iscsi_ip_prefix=192.168.1
–volume_manager	default: 'nova.volume.manager.VolumeManager'	String value; Manager to use for nova-volume
–volume_name_template	default: 'volume-%08x'	String value; Template string to be used to generate volume names
–volume_topic	default: 'volume'	String value; Name of the topic that volume nodes listen on

10. 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 a reference implementation of a Django site that provides web-based interactions with the OpenStack Compute cloud controller. For more information about the Openstack-Dashboard project, please visit: http://launchpad.net/openstack-dashboard. These instructions are for a test deployment of an OpenStack Dashboard. They configure your dashboard to use the default Django server. To create a more robust, production-ready installation, you would configure this with an Apache web server.

System Requirements for the Dashboard

You should have a running OpenStack Compute installation with the Keystone module enabled for identity management. Follow these general instructions for installing Identity Management. For a good all-in-one Nova/Glance/Keystone installation there is the devstack project.

The dashboard needs to be installed on the node that can contact the Keystone service.

You should know the URL of your Identity endpoint and the Compute endpoint.

You must know the credentials of a valid Keystone tenant.

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 building the dashboard.

1. Get the source for the openstack-dashboard project.

2. Configure the openstack-dashboard.

3. Install.

4. Run the server that starts the dashboard.

Before you begin, you must have git installed.

Create a source directory to house the project:

```
mkdir src
cd src
```

Next, get the openstack-dashboard project, which provides all the look and feel for the OpenStack Dashboard.

```
git clone https://github.com/4P/openstack-dashboard
```

You should now have a directory called openstack-dashboard, which contains the OpenStack Dashboard application.

Configure Openstack-Dashboard

Now you can configure the dashboard application. The first step in configuring the application is to create your local_settings.py file. An example is provided that you can copy to local_settings.py and then modify for your environment.

```
cd openstack-dashboard/openstack-dashboard
cp local_settings.py.example local/local_settings.py
vi local_settings.py
```

In the new copy of the local_settings.py file, change these important options:

*   OPENSTACK_ADMIN_TOKEN : Token for Keystone endpoint.

*   OPENSTACK_KEYSTONE_URL : URL for the Keystone endpoint.

Keystone Configuration (required)

The admin token can be generated by executing something like the following using the keystone-manage command on the Keystone host:

```
 keystone-manage token add 999888777666 admin admin 2015-02-05T00:00
```

To use this token you would add the following to local_settings.py:

```
OPENSTACK_ADMIN_TOKEN = "999888777666"
```

The Keystone endpoint setting takes the following form:

```
OPENSTACK_KEYSTONE_URL = "http://mykeystoneurl:5000/v2.0/".
```

Object Storage Configuration (optional)

If a Swift endpoint is available and configured in the Keystone service catalog turning on the Swift UI is as simple as adding the following to local_settings.py:

```
SWIFT_ENABLED = True
```

Quantum Configuration (optional)

Quantum currently requires the following settings:

```
QUANTUM_ENABLED = True
QUANTUM_URL = '127.0.0.1'
QUANTUM_PORT = '9696'
QUANTUM_TENANT = '1234'
QUANTUM_CLIENT_VERSION='0.1'
```

Install the Dashboard

After Dashboard has been configured install the Dashboard virtual environment using the terminal commands below:

Note

Note: the instructions below are for Ubuntu, however, setuptools can be installed on a wide variety of platforms: http://pypi.python.org/pypi/setuptools

```
$ apt-get install -y python-setuptools
            $ sudo easy_install virtualenv
            $ python tools/install_venv.py
```

Installing the virtual environment will take some time depending on download speeds.

Run the Server

Dashboard is run using the standard Django manage.py script from the context of the virtual environment. Run the server on a high port value so that you can validate the installation.

```
tools/with_venv.sh dashboard/manage.py runserver 0.0.0.0:8000
```

Make sure that your firewall isn't blocking TCP/8000 and just point your browser at this server on port 8000. If you are running the server on the same machine as your browser, this would be "http://localhost:8000".
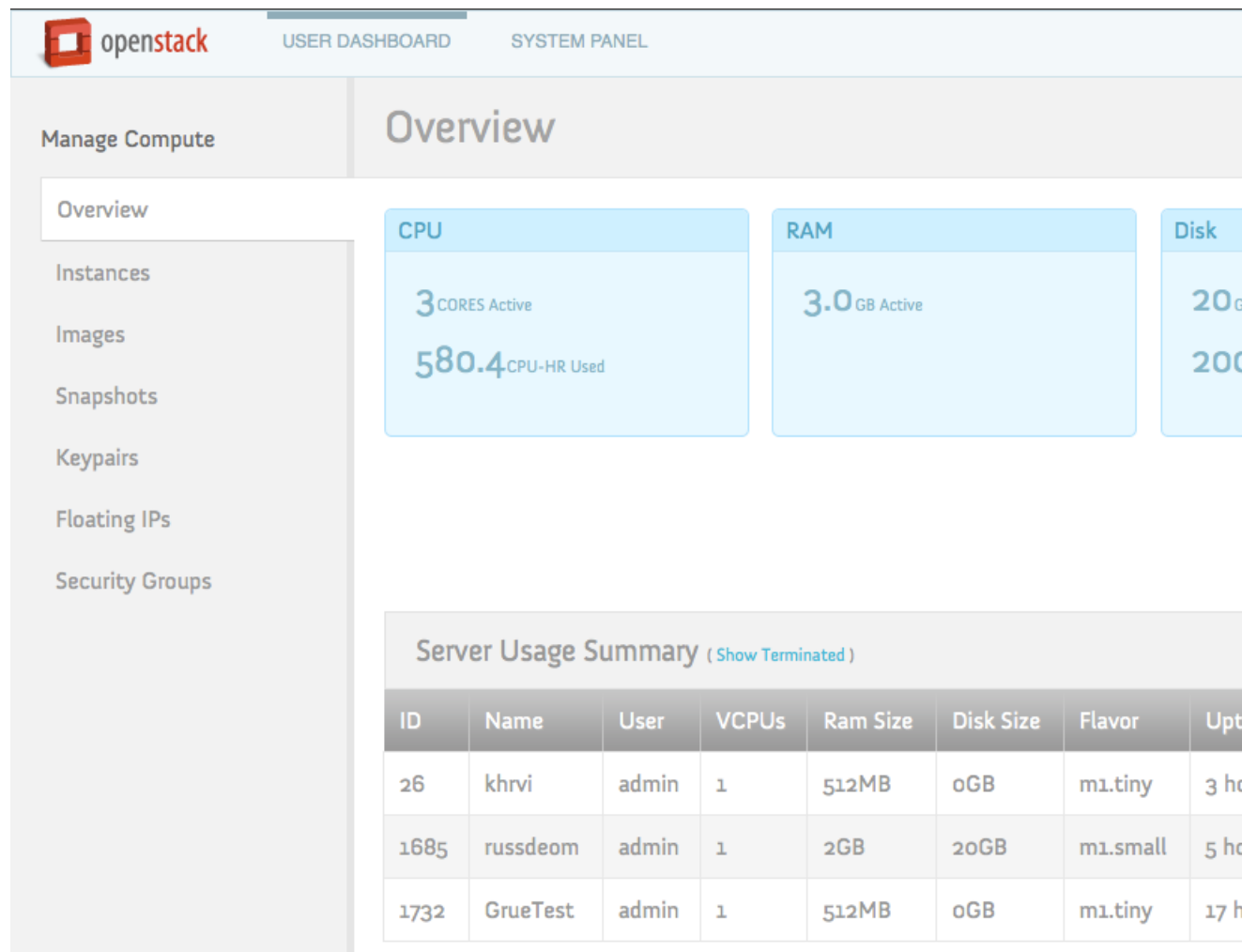
Getting Started with the VNC Proxy

The VNC Proxy is an OpenStack component that allows users of Nova to access their instances through a websocket enabled browser (like Google Chrome 4.0). See http://caniuse.com/#search=websocket for a reference list of supported web browsers.

A VNC Connection works like so:

• User connects over an API and gets a URL like http://ip:port/?token=xyz

• User pastes URL in browser

• Browser connects to VNC Proxy though a websocket enabled client like noVNC

• VNC Proxy authorizes users token, maps the token to a host and port of an instance's VNC server

• VNC Proxy initiates connection to VNC server, and continues proxying until the session ends

Configuring the VNC Proxy

The nova-vncproxy requires a websocket enabled html client to work properly. At this time, the only tested client is a slightly modified fork of noVNC, which you can at find http:// github.com/openstack/noVNC.git

The noVNC tool must be in the location specified by –vncproxy_wwwroot, which defaults to /var/lib/nova/noVNC. nova-vncproxy will fail to launch until this code is properly installed.

By default, nova-vncproxy binds 0.0.0.0:6080. This can be configured with:

• –vncproxy_port=[port]

• –vncproxy_host=[host]

Enabling VNC Consoles in Nova

At the moment, VNC support is supported only when using libvirt. To enable VNC Console, configure the following flags in the nova.conf file:

• –vnc_console_proxy_url=http://[proxy_host]:[proxy_port] - proxy_port defaults to 6080. This URL must point to nova-vncproxy

• –vnc_enabled=[True|False] - defaults to True. If this flag is not set your instances will launch without VNC support.

Getting an Instance's VNC Console

You can access an instance's VNC Console URL in the following methods:

• Using the direct api: eg: 'stack –user=admin –project=admin compute get_vnc_console instance_id=1'

• Support for Dashboard, and the Openstack API will be forthcoming

At the moment, VNC Consoles are only supported through the web browser, but more general VNC support is in the works.

11. 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 OpenStack Compute already installed on Ubuntu 10.04. You can tell OpenStack Compute is installed by running "sudo nova-manage service list" to ensure it is installed and the necessary services are running and ready. You should see a set of nova- services in a response, and they should have a sideways smiley face in each row, indicating they're running. You should run the tutorial as a root user or a user with sudo access.

If you haven't installed OpenStack Compute yet, you can 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 to get started.

We'll go through this tutorial in parts:

• Setting up a user, project, and network for this 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 the Cloud Infrastructure

In this part, we'll get the networking layer set up based on what we think most networks would work like. We'll also create a user and a project to house our cloud and its network. Onward, brave cloud pioneers!

Configuring the network

Ideally on large OpenStack Compute deployments, each project is in a protected network segment. Our project in this case is a LAMP stack running Wordpress with the Memcached plugin for added database efficiency. So we need a public IP address for the Wordpress server but we can use flat networking for this. Here's how you set those network settings.

Usually networking is set in nova.conf, but VLAN-based networking with DHCP is the default setting when no network manager is defined in nova.conf. To check this network setting, open your nova.conf, typically in /etc/nova/nova.conf and look for -network_manager. The possible options are:

- -network_manager=nova.network.manager.FlatManager for a simple, no-VLAN networking type,

- -network_manager=nova.network.manager.FlatDHCPManager for flat networking with a built-in DHCP server,

- -network_manager= nova.network.manager.VlanManager, which is the most tested in production but requires network hardware with VLAN tagging.

Here is an example nova.conf for a single node installation of OpenStack Compute.

```
# Sets the network type
--network_manager=nova.network.manager.FlatManager
# Sets whether to use IPV6 addresses
--use_ipv6=false
# DHCP bridge information
--dhcpbridge_flagfile=/etc/nova/nova.conf
--dhcpbridge=nova-dhcpbridge
--flat_network_bridge=br100
--logdir=/var/log/nova
# Top-level directory for maintaining nova's state
--state_path=/var/lib/nova
# These indicate where nova-api services are installed
--s3_host=184.106.239.134
--rabbit_host=184.106.239.134
--ec2_api=184.106.239.134
--ec2_url=http://184.106.239.134:8773/services/Cloud
# Block of IP addresses that are fixed IPs
--fixed_range=192.168.0.0/12
# Number of addresses in each private subnet
--network_size=24
# FlatDHCP bridges to this interface if set, be very careful setting it on an
 interface that does not already have an IP associated with it
--flat_interface=eth0
# Public IP of the server running nova-network, when instances without a
 floating IP hit the internet, traffic is snatted to this IP
--routing_source_ip=184.106.239.134
# Not required, but handy for debugging
--verbose
# Tells nova where to connect for database
--sql_connection=mysql://nova:notnova@184.106.239.134/nova
```

Now that we know the networking configuration, let's set up the network for our project. With Flat DHCP, the host running nova-network acts as the gateway to the virtual nodes, so ideally this will have a public IP address for our tutorial. Be careful when setting up – flat_interface in nova.conf, 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. Also the –flat_network_bridge is now required.

For this tutorial, we set a 24 value for network_size, the number of addresses in each private subnet, since that falls inside the /12 CIDR-notated range that's set in 'fixed-range' in nova.conf. We probably won't use that many at first, but it's good to have the room to scale.

Currently, there can only be one network set in nova.conf. When you issue the nova-manage network create command, it uses the settings in the nova.conf flag file. From the –fixed_range setting, iptables are set. Those iptables are regenerated each time the nova-network service restarts, also.

> **Note**
>
> 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.o but that scenario shouldn't happen for this tutorial.

Run this command as root or sudo:

```
nova-manage network create public 192.168.3.0/12 1 256
```

On running this command, entries are made in the 'networks' and 'fixed_ips' table in the nova database. 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 is marked as bridged automatically based on the type of network manager selected.

Next you want to integrate this network bridge, named br100, into your network. A bridge connects two Ethernet segments together.

Ensure the Database is Up-to-date

The first command you run using nova-manage is one called db sync, which ensures that your database is updated. You must run this as root.

```
nova-manage db sync
```

Creating a user

OpenStack Compute can run many projects for many users, so for our tutorial we'll create a user and project just for this scenario.

We control the actions a user can take through roles, such as admin for Administrator who has complete system access, itsec for IT Security, netadmin for Network Administrator, and so on.

In addition to these roles controlling access to the Eucalyptus API, credentials are supplied and bundled by OpenStack compute in a zip file when you create a project. The user accessing the cloud infrastructure through ec2 commands are given an access and secret key through the project itself. Let's create a user that has the access we want for this project.

To add an admin user named cloudypants, use:

```
nova-manage user admin cloudypants
```

Creating a project and related credentials

Next we'll create the project, which in turn gives you certifications in a zip file.

Enter this command to create a project named wpscales as the admin user, cloudypants, that you created above.

```
nova-manage project create wpscales cloudypants
```

Great, now you have a project that is set apart from the rest of the clouds you might control with OpenStack Compute. Now you need to give the user some credentials so they can run commands for the instances with in that project's cloud.

These are the certs you will use to launch instances, bundle images, and all the other assorted API and command-line functions.

First, we'll create a directory that'll house these credentials, in this case in the root directory. You need to sudo here or save this to your own directory with 'mkdir -p ~/creds' so that the credentials match the user and are stored in their home.

```
mkdir –p /root/creds
```

Now, run nova-manage to create a zip file for your project called wpscales with the user cloudypants (the admin user we created previously).

```
sudo nova-manage project zipfile wpscales cloudypants /root/creds/
novacreds.zip
```

Next, you can unzip novacreds.zip in your home directory, and add these credentials to your environment.

```
unzip /root/creds/novacreds.zip -d /root/creds/
```

Sending that information and sourcing it as part of your .bashrc file remembers those credentials for next time.

```
cat /root/creds/novarc >> ~/.bashrc
source ~/.bashrc
```

Okay, you've created the basic scaffolding for your cloud so that you can get some images and run instances. Onward to Part II!

Part II: Getting Virtual Machines to Run the Virtual Servers

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.

For this tutorial, we've created an image that you can download that allows the networking you need to run web applications and so forth. In order to use it with the OpenStack Compute cloud, you download the image, then use uec-publish-tarball to publish it.

Here are the commands to get your virtual image. Be aware that the download of the compressed file may take a few minutes.

```
image="ubuntu1010-UEC-localuser-image.tar.gz"
wget http://c0179148.cdn1.cloudfiles.rackspacecloud.com/
ubuntu1010-UEC-localuser-image.tar.gz
uec-publish-tarball $image wpbucket x86_64
```

What you'll get in return from this command is three references: emi, eri and eki. These are acronyms - emi stands for eucalyptus machine image, eri stands for eucalyptus ramdisk

image, and eki stands for eucalyptus kernal image. Amazon has similar references for their images - ami, ari, and aki.

You need to use the emi value when you run the instance. These look something like "ami-zqkyh9th" - basically a unique identifier.

Okay, now that you have your image and it's published, realize that it has to be decompressed before you can launch an image from it. We can realize what state an image is in using the 'euca-describe-instances' command. Basically, run:

```
euca-describe-instances
```

and look for the state in the text that returns. You can also use euca-describe-images to ensure the image is untarred. Wait until the state shows "available" so that you know the instances is ready to roll.

Part III: Installing the Needed Software for the Web-Scale Scenario

Once that state is "available" you can enter this command, which will use your credentials to start up the instance with the identifier you got by publishing the image.

```
emi=ami-zqkyh9th
euca-run-instances $emi -k mykey -t m1.tiny
```

Now you can look at the state of the running instances by using euca-describe-instances again. The instance will go 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.

Basically launch a terminal window from any computer, and enter:

```
ssh -i mykey 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.

12. 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 flags for the nova.log

module. The flags 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 flag 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.