OpenStack
Operations Guide

Tom Fifield, Diane Fleming, Anne Gentle, Lorin Hochstein,
Jonathan Proulx, Everett Toews, and Joe Topjian

Facilitator: Adam Hyde

OpenStack Operations Guide

Contents

Introduction 1
Why We Wrote This Book 3
How to Work on This Book 7

Architecture 9
Provisioning and Deploying 11
Cloud Controller Design 15
Scaling 21
Compute Nodes 29
Decisions for Storage 35
Network Design 43
Example Architecture 49
What Next 55

Operations 57
Lay of the Land 59
Managing Projects and Users 71
User Facing Operations 83
Maintenance, Failures, and Debugging 103
Network Troubleshooting 121
Logging and Monitoring 131
Backup and Recovery 145
Customize 149
Upstream OpenStack 165

Appendices 173
Use Cases 175
Tales From the Cryp^H^H^H^H Cloud 179
Recommended Reading 189
Glossary 191

Introduction

Why We Wrote This Book

How to Work on This Book

Why We Wrote This Book

Your authors cobbled this book together in five days during February 2013, fueled by caffeine and the best take-out food Austin, Texas could offer. We wrote this book because we have deployed and maintained OpenStack clouds for at least a year, and wanted to be able to distribute this knowledge for others to learn from. After months of being the point person for an OpenStack cloud, we also wanted to have a document to hand to our system administrators so they'd know how to operate it daily — both re-actively and proactively. We wanted to provide more detailed technical information about the decisions deployers made along the way.

This guide has two goals. The first goal is to provide guidance for designing and creating an architecture for your first non-trivial OpenStack cloud. After reading this guide, you should have a better sense of what questions to ask and how to organize your compute, networking, storage resources and the associated software packages. The second goal is to provide you with assistance on the day-to-day tasks involved in administering a cloud.

The book was written in a Book Sprint. This is a facilitated rapid development production method for books. For more information see the Book Sprint site (http://www.booksprints.net).

Why OpenStack

OpenStack enables you to build an Infrastructure as a Service (IaaS) cloud. If you are looking to build a cloud with the resources you have, OpenStack is an excellent choice. It's built from open source, and is designed to run on commodity hardware. In fact, many operators build an initial proof-of-concept OpenStack cloud by scrounging together whatever used servers and networking gear that they can get their hands on. OpenStack was also designed for scalability, so you can easily add new compute and storage resources to grow your cloud over time. Organizations such as HP and Rackspace have built massive public clouds on top of OpenStack.

If you're new to OpenStack, you'll soon discover that it isn't a software package that can simply be run as-is. Instead, it's a system that lets you integrate a number of different technologies in order to construct a cloud. This approach provides an you with a large degree of flexibility, but the number of options can be bewildering at first.

Appreciation and Acknowledgements

The OpenStack Foundation supported the creation of this book with plane tickets to Austin, lodging (including one adventurous evening without power after a windstorm), and delicious food. For about USD $10,000, we could collaborate intensively for a week in the same room at Rackspace's Austin office. The authors are all members of the OpenStack Foundation and you can join also by going to the Foundation web site at http://openstack.org/join.

On the first day we filled white boards with colorful sticky notes to start to shape this nebulous book about how to architect and operate clouds.

We wrote furiously from our own experiences and bounced ideas between us. At regular intervals we reviewed the shape and organization of the book and further molded it, leading to what you see today.

The team includes:

Diane Fleming - Diane works on the OpenStack API documentation tirelessly and was willing to help out wherever she could on this project.

Tom Fifield - After learning about scalability in computing from particle physics experiments like ATLAS at the LHC, Tom has been working on OpenStack clouds in production to support the Australian public research sector. He lives in Melbourne, Australia and works on OpenStack documentation in his spare time.

Anne Gentle - Anne is the documentation coordinator for OpenStack and also served as an individual contributor to the Google Doc Summit in 2011, working with the Open Street Maps team. Anne has worked on doc sprints in the past with FLOSS Manuals' Adam Hyde facilitating. Anne lives in Austin, Texas.

Lorin Hochstein - Lorin is an academic turned software developer-slash-operator. He currently works as the Lead Architect for Cloud Services at Nimbis Services where he deploys OpenStack for technical computing applications. He has been working with OpenStack since the Cactus release. Previously, he worked on high-performance computing extensions for OpenStack at University of Southern California's Information Sciences Institute (USC-ISI).

Adam Hyde - Adam facilitated this Book Sprint. He also founded the Book Sprint methodology and is the most experienced Book Sprint facilitator around. See http://www.booksprints.net/ for more information. Adam founded FLOSS Manuals—a community of some 3,000 individuals developing Free Manuals about Free Software. He is also the founder and project manager for Booktype, an open source project for writing, editing, and publishing books online and in print.

Jonathan Proulx - Jon has been piloting an OpenStack cloud as a senior system administrator at the MIT Computer Science and Artificial Intelligence Lab for his researchers to have as much computing power as they need. He started contributing to OpenStack documentation and reviewing the documentation so that he could accelerate his learning.

Everett Toews - Everett Toews is a Developer Advocate at Rackspace making OpenStack and the Rackspace Cloud easy to use. Sometimes developer, sometimes advocate, and sometimes operator. He's built web applications, taught workshops, given presentations around the world, and deployed OpenStack for production use by academia and business.

Joe Topjian - Joe has designed and deployed several clouds at Cybera, where, as a non-profit, they are building e-infrastructure to support entrepreneurs and local researchers in Alberta, Canada. He also actively maintains and operates these clouds which has generated a wealth of troubleshooting skills for cloud environments.

We all want to acknowledge our excellent host Rackers at Rackspace in Austin. Emma Richards of Rackspace Guest Relations took excellent care of our lunch orders and even set aside a pile of sticky notes that had fallen off the walls. Betsy Hagemeier, a Fanatical Executive Assistant, took care of a room reshuffle and helped us settle in for the week. The Real Estate team at Rackspace

in Austin, also known as "The Victors," were super responsive. Adam Powell in Racker IT supplied us with bandwidth each day and second monitors for those of us needing more screens. The cover designer, Chris Duncan, turned the design around from a Friday inquiry email (you want to do what by when?) to a Friday delivery email, including source files and open source fonts. On Wednesday night we had a fun happy hour with the Austin OpenStack Meetup group and Racker Katie Schmidt took great care of our group. We couldn't have pulled it off with out so much supportive help and encouragement.

We also had some excellent input from outside of the room. Tim Bell from CERN gave us feedback on the outline before we started and reviewed it mid-week. Sébastien Han has written excellent blogs and generously gave his permission for re-use. Oisin Feeley read it, made some edits, and provided emailed feedback right when we asked.

What You Need to Know

We assume you have experience with Linux systems administration tasks and are comfortable administering multiple Linux machines. In particular, this guide assumes knowledge of the Ubuntu distribution of Linux. We also assume you have some experience with SQL databases, since you'll need to install and maintain a MySQL database, and occasionally run SQL queries against it. Finally, we assume some familiarity with virtualization.

We also assume you have experience configuring Linux machines for networking: one of the most complex aspects of an OpenStack cloud is the networking configuration. You should already be familiar with concepts such as DHCP, Linux bridges, VLANs, and iptables. You'll also need to have access to someone with network hardware expertise who can configure the switches and routers you'll be using in your OpenStack cloud.

How to Work on This Book

The genesis of this book was an in-person event, but now that the book is in your hands we want you to be able to contribute to it. OpenStack documentation follows the coding principles of iterative work, with bug logging, investigating, and fixing.

If you see a bug in this book or want to work on a section, you can edit it immediately by going to the OpenStack Booktype site (http://openstack.booktype.pro), creating a user account, and editing the section as you see fit. The book is located at OpenStack Operations Guide (http://openstack.booktype.pro/openstack-operations-guide/).

A member of the OpenStack doc-core team publishes the entire book periodically. We plan to have events at OpenStack Summits to work in-person on this book.

If you can't fix it immediately or aren't sure it's really a doc bug, log a bug at OpenStack Manuals (http://bugs.launchpad.net/openstack-manuals) and indicate the doc bug is in the Operations Manual by tagging it with "ops-guide" under Extra options. You can assign yourself to work on it once you determine a good way forward. Also, a member of the OpenStack doc-core team can triage the doc bug. Learn more about the documentation workflow at the Documentation How To (http://wiki.openstack.org/Documentation/HowTo).

Architecture

- Provisioning and Deploying
- Cloud Controller Design
- Scaling
- Compute Nodes
- Decisions for Storage
- Network Design
- Example Architecture
- What Next

Provisioning and Deploying

A critical part of a cloud's scalability is the amount of effort that it takes to run your cloud. To minimize the operational cost of running your cloud, set up and use an automated deployment and configuration infrastructure. This includes systems to automatically install the operating system's initial configuration and to later coordinate the configuration of all services automatically and centrally, which reduces both manual effort and chance for error.

Automated Deployment

An automated deployment system installs and configures operating systems on new servers, without intervention, after the absolute minimum amount of manual work (physical racking, MAC to IP assignment, power configuration, and so on). Typically solutions rely on wrappers around PXE boot and TFTP servers for the basic operating system install, then hand off to an automated configuration management system.

Ubuntu and Red Hat Linux both include mechanisms for configuring the operating system, including preseed and kickstart, that you can use after a network boot. Typically these are used to bootstrap an automated configuration system. Alternatively, you can use an image-based approach for deploying the operating system, such as systemimager. You can use both approaches with a virtualized infrastructure, such as when you run VMs to separate your control services and physical infrastructure.

When you create a deployment plan, focus on a few vital areas because they are very hard to modify post-deployment.

Disk Partitioning and RAID

At the very base of any operating system are the hard drives on which the OS is installed.

You must complete the following configurations on the server's hard drives:

Partitioning
Adding to a RAID array

The simplest option is to use one hard drive with two partitions:

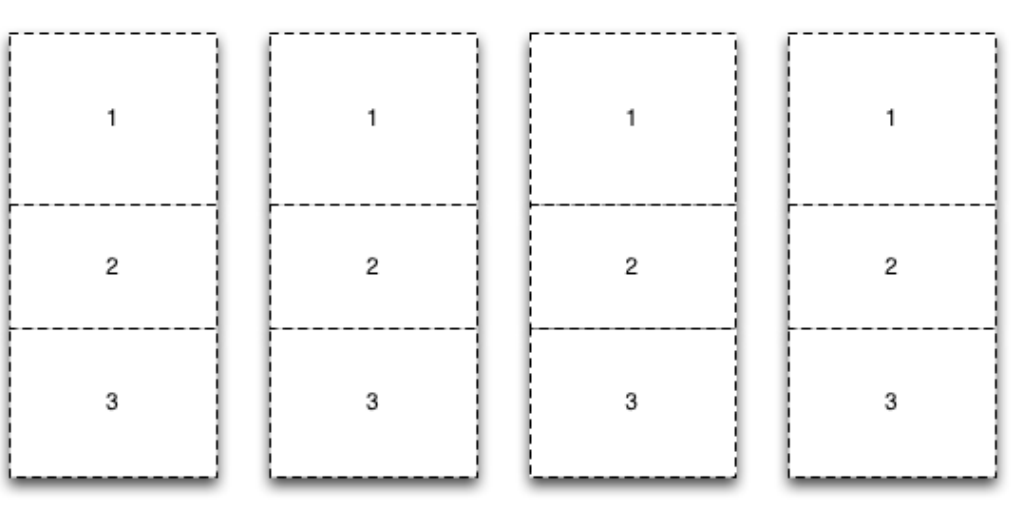
File system
Swap space

RAID is not used in this setup.

Note: This option is not recommended for production because if the hard drive fails, that entire server is down. Instead, we recommend that you use more than one disk. The number of disks determine what types of RAID arrays to build.

We recommend that you choose one of the following multiple disk options:

> **Option 1:** Partition all drives in the same way in a horizontal fashion, as shown in the following diagram:



With this option, you can assign different partitions to different RAID arrays. You can allocate partition 1 of disk one and two to the `/boot` partition mirror. You can make partition 2 of all disks the root partition mirror. You can use partition 3 of all disks for a `cinder-volumes` LVM partition running on a RAID 10 array.

While you might end up with unused partitions, such as partition 1 in disk three and four of this example, it allows for maximum utilization off disk space. I/O performance might be an issue due to all disks being used for all tasks.

> **Option 2:** Add all raw disks to one large RAID array, either hardware or software based. You can partition this large array with the boot, root, swap, and LVM areas. This option is simple to implement and uses all partitions. However, disk I/O might suffer.

Option 3: Dedicate entire disks to certain partitions. For example, you could allocate disk one and two entirely to the boot, root, and swap partitions under a RAID 1 mirror. Then, allocate disk 3 and 4 entirely to the LVM partition, also under a RAID 1 mirror. Disk I/O should be better because I/O is focused on dedicated tasks. However, the LVM partition will be much smaller.

As with most architecture choices, the right answer depends on your environment.

Network Configuration

Network configuration is a very large topic that spans multiple areas of this book. For now, make sure that your servers can PXE boot and successfully communicate with the deployment server.

For example, you usually cannot configure NICs for VLANs when PXE booting. Additionally, you usually cannot PXE boot with bonded NICs. If you run into this scenario, consider using a simple 1 GB switch in a private network that only your cloud will communicate on.

Automated Configuration

The purpose of automatic configuration management is to establish and maintain the consistency of a system with no human intervention. You want to maintain consistency in your deployments so you can have the same cloud every time, repeatably. Proper use of automatic configuration management tools ensures that components of the cloud systems are in particular states, in addition to simplifying deployment, and configuration change propagation. These tools also make it possible to test and roll back changes, as they are fully repeatable. Conveniently, a large body of work has been done by the OpenStack community in this space. Puppet – a configuration management tool – even provides official modules for OpenStack.

An integral part of a configuration management system is the items that it controls. You should carefully consider all of the items that you want, or do not want, to be automatically managed.

Remote Management

In our experience, most operators don't sit right next to the servers running the cloud, and many don't necessarily enjoy visiting the data center. OpenStack should be entirely remotely configurable, but sometimes not everything goes according to plan. In this instance, having an out-of-band access into nodes running OpenStack components, is a boon. The IPMI protocol is the de-facto standard here, and acquiring hardware that supports it is highly recommended to achieve that lights-out datacentre aim.

In addition, consider remote power control as well. While IPMI usually controls the server's power state, having remote access to the PDU that the server is plugged into can really be useful for situations when everything seems wedged.

Cloud Controller Design

OpenStack is designed to be massively horizontally scalable, which allows all services to be distributed widely. However, to simplify this guide we have decided to discuss services of a more central nature using the concept of a single *cloud controller* (see the **Example Architecture** chapter for more details on the overall architecture). As described in this guide, the cloud controller is a single node that hosts the databases, the message queue service, the authentication and authorization service, the image management service, user dashboard and API endpoints.

The cloud controller provides the central management system for multi-node OpenStack deployments. Typically the cloud controller manages authentication and sends messaging to all the systems through a message queue. For our example, the cloud controller has a collection of `nova-*` components that represent the global state of the cloud, talks to services such as authentication, maintains information about the cloud in a database, communicates to all compute nodes and storage workers through a queue, and provides API access. Each service running on a designated cloud controller may be broken out into separate nodes for scalability or availability.

Hardware Considerations

A cloud controller's hardware can be the same as a compute node, though you may want to further specify based on the size and type of cloud that you run. It's also possible to use virtual machines for all or some of the services that the cloud controller manages, such as the message queuing. In this guide, we assume that all services are running directly on the cloud controller.

To size the server correctly, and determine whether to virtualize any part of it, you should estimate:

how many instances you expect to run
how many compute nodes you have
how many users will access the compute or storage services
whether users are interacting with your cloud through the REST API or through the Dashboard
whether users authenticate against an external system (e.g., LDAP, Active Directory)
how long you expect single instances to remain running

Consideration	Ramification
How many instances will run at once?	Size your database server accordingly, and scale out beyond one cloud controller if many instances will be reporting status at the same time and scheduling where a new instance starts up needs computing power.
How many compute nodes will run at once?	Ensure your messaging queue is handling the requests well and sized accordingly.
How many users will access the API?	If many users will be making multiple requests, ensure the CPU load for the cloud controller can handle it.
How many users will access the Dashboard?	The Dashboard makes many requests, even more than the API access, so add even more CPU if your Dashboard is the main interface for your users.
How many `nova-api` services do you run at once for your cloud?	You need to size the controller with a core per service.
How long does a single instance run?	Starting instances and deleting instances is demanding on the compute node but also demanding on the controller node because of all the API queries and scheduling needs.
Does your authentication system also verify externally?	Ensure network connectivity between the cloud controller and external authentication system are good and that the cloud controller has the CPU power to keep up with requests.

Separation of services

While our example contains all central services in a single location, it is possible and indeed often a good idea to separate services onto different physical servers. The following is a list of deployment scenarios we've seen, and their justifications.

Run `glance-*` servers on the `swift-proxy` server	This deployment felt the spare I/O on the Object Storage proxy server was sufficient, and the Image Delivery portion of Glance benefited from being on physical hardware and having good connectivity to the Object Storage backend it was using.
Run a central dedicated database server	This deployment made a central dedicated server to provide the databases for all services. This simplified operations by isolating database server updates, and allowed for the simple creation of slave database servers for failover.
Run one VM per service	This deployment ran central services on a set of servers running KVM. A dedicated VM was created for each service (nova-scheduler, rabbitmq, database etc). This assisted the deployment with scaling as they could tune the resources given to each virtual machine based on the load they received (something that was not well undertood during installation).
Use an external load balancer	This deployment had an expensive hardware load balancer in their organisation. They ran multiple `nova-api` and `swift-proxy` servers on different physical servers and used the load balancer to switch between them.

One choice that always comes up is whether to virtualise or not. Some services, such as nova-compute, swift-proxy and swift-object servers should not be virtualised. However, control servers can often be happily virtualised - the performance penalty can usually be offset by simply running more of the service.

Database

Most OpenStack Compute central services, and currently also the nova-compute nodes, use the database for stateful information. The loss of access to this ability will lead to errors. As a result, we recommend that you cluster your databases in some way to make them failure tolerant.

Message Queue

Most OpenStack Compute services communicate with each other using the Message Queue. In general, if the message queue fails or becomes inaccessible, the cluster grinds to a halt and ends up in a "read only" state, with information stuck at the point where the last message was sent. Accordingly, we recommend that you cluster the message queue - and RabbitMQ has in-build abilities to do this.

API

All public access, whether it is direct, via a command line client, or through the web-based dashboard uses the API service. An API reference can be found at http://api.openstack.org/.

You need to make a choice as to whether you want to support the Amazon EC2 compatibility APIs, or just the OpenStack APIs. One issue you may encounter when running both APIs is an inconsistent experience when referring to images and instances. For example, the EC2 API refers to instances using IDs that contain hexadecimal whereas the OpenStack API uses names and digits. Similarly, the EC2 API tends to rely on DNS aliases for contacting virtual machines, as opposed to OpenStack which typically lists IP addresses. If OpenStack is not setup in the right way, it is simple to have scenarios where users are unable to contact their instances due to only having an incorrect DNS alias. Despite this, EC2 compatibility can assist users migrating to your cloud.

Like databases and message queues, having more than one API server is a good thing. Traditional HTTP load balancing techniques can be used to achieve a highly available `nova-api` service.

Extensions

The API Specifications (http://docs.openstack.org/api/api-specs.html) define the core actions, capabilities, and media-types of the OpenStack API. A client can always depend on the availability of this core API and implementers are always required to support it in its entirety. Requiring strict adherence to the core API allows clients to rely upon a minimal level of functionality when interacting with multiple implementations of the same API.

The OpenStack Compute API is extensible. An extension adds capabilities to an API beyond those defined in the core. The introduction of new features, MIME types, actions, states, headers, parameters, and resources can all be accomplished by means of extensions to the core API. This allows the introduction of new features in the API without requiring a version change and allows the introduction of vendor-specific niche functionality.

Scheduler

Fitting various sized virtual machines (different *flavors*) into different sized physical nova-compute nodes is a challenging problem - researched generically in Computer Science as a packing problem. There are various techniques for handling this problem, one of which is to have flavor sizes scale linearly, be a proportional fraction of your physical node capacity, though solving this problem is out of the scope of this book. To support your scheduling choices, OpenStack Compute provides several different types of scheduling drivers, a full discussion of which is found in the reference manual (http://docs.openstack.org/folsom/openstack-compute/admin/content/ch_scheduling.html).

For availability purposes, or for very large or high-schedule frequency installations, you should consider running multiple nova-scheduler services. No special load balancing is required, as the nova-scheduler communicates entirely using the message queue.

Images

The OpenStack Image Catalog and Delivery service consists of two parts - `glance-api` and `glance-registry`. The former is responsible for the delivery of images and the compute node uses it to download images from the backend. The latter maintains the metadata information associated with virtual machine images and requires a database.

The `glance-api` part is an abstraction layer that allows a choice of backend. Currently, it supports:

OpenStack Object Storage, which allows the storage of images as objects.
File System, which uses any traditional file system for storing the images as files.
S3, which allows the fetching of images from Amazon S3 (no images can be written using this mode).
HTTP, which allows the fetching of images from a web server (no images can be written using this mode).

If you have an OpenStack Object Storage service, we recommend using this as a scalable place to store your images. Failing that, a filesystem with sufficient performance is the only real option - unless you do not need the ability to upload new images via OpenStack.

Dashboard

The OpenStack Dashboard is implemented as a Python web application that runs in Apache `httpd`. Therefore, you may treat it the same as any other web application, provided it can reach the API servers (including their admin endpoints) over the network.

Authentication and Authorization

The concepts supporting OpenStack's authentication and authorization are derived from well understood and widely used systems of a similar nature. Users have credentials they can use to authenticate, and they can be a member of one or more groups (known as projects or tenants interchangeably). For example a cloud administrator might be able to list all instances in the cloud, whereas a user can only see those in their current group. Resources quotas, such as the number of cores that can be used, disk space, and so on, are associated with a project.

The OpenStack Identity Service (Keystone) is the point that provides the authentication decisions and user attribute information, which is then used by the other OpenStack services to perform authorization. Policy is set in files known as policy.json - see the **User Support chapter** for information on how to configure these.

The Identity Service supports different plugins for backend authentication decisions, and storing information. These range from pure storage choices to external systems and currently include:

in-memory Key-Value Store
SQL database
PAM
LDAP

Many deployments use the SQL database, however LDAP is also a popular choice for those with existing authentication infrastructure that needs to be integrated.

Network Considerations

Since the cloud controller handles so many different services, it is important that it can handle the amount of traffic that will be hitting it. For example, if you choose to host the OpenStack Imaging Service on the cloud controller, the cloud controller should be able to support the transferring of the images at an acceptable speed.

As another example, if you choose to use single-host networking where the cloud controller will be the network gateway for all instances, then the Cloud Controller must be able to support the total amount of traffic going to and from the public Internet that you will expect from your cloud.

It is recommended to use a fast NIC, such as 10gb. You can also choose to use two 10gb NICs and bond them together. While you might not be able to get a full bonded 20gb speed, different transmission streams will utilize different NICs. For example, the Cloud Controller is transferring two images, each image will use a different NIC and get a full 10gb of bandwidth.

Scaling

If your cloud is successful, you will eventually need to add resources to meet the increasing demand. OpenStack is designed to be horizontally scalable. Rather than switching to larger servers, you procure more servers. Ideally, you scale out and load balance among functionally-identical services.

The Starting Point

Determining the scalability of your cloud and how to improve it is an exercise with many variables to balance. No one solution meets everyone's scalability aims. However, it is helpful to track a number of metrics.

The starting point for most is the core count of your cloud. By applying some ratios, you can gather information about the number of virtual machines (VMs) you expect to run ((overcommit fraction × cores) / virtual cores per instance), how much storage will be required (flavor disk size × number of instances). You can then use these ratios to determine how much additional infrastructure you will require to support your cloud.

The default OpenStack flavors are:

Name	Virtual cores	Memory	Disk	Ephemeral
m1.tiny	1	512 MB	0 GB	0 GB
m1.small	1	2 GB	10 GB	20 GB
m1.medium	2	4 GB	10 GB	40 GB
m1.large	4	8 GB	10 GB	80 GB
m1.xlarge	8	16 GB	10 GB	160 GB

If we make the following assumptions:

- 200 physical cores
- Most instances are size m1.medium (2 virtual cores, 50 GB of storage)
- Default CPU over-commit ratio (`cpu_allocation_ratio` in nova.conf) of 16:1

Then this would support (200 / 2) × 16 = 1600 VM instances and requires 80 TB of storage for `/var/lib/nova/instances`.

However, you need more than the core count alone to estimate the load that the API services, database servers, and queue servers are likely to encounter. You must also consider the usage patterns of your cloud. As a specific example, compare a cloud that supports a managed web hosting platform with one running integration tests for a development project that creates one VM per code commit. In the former, the heavy work of creating a VM happens only every few months, whereas the latter will put constant heavy load on the cloud controller. You must consider your average VM lifetime, as a larger number generally means less load on the cloud controller.

Aside from the the creation and termination of VMs, you must consider the impact of users accessing the service — particularly on nova-api and its associated database. Listing instances garners a great deal of information and, given the frequency with which users run this operation, a cloud with a large number of users can increase the load significantly. This can even occur without their knowledge — leaving the OpenStack Dashboard instances tab open in the browser refreshes the list of VMs every 30 seconds.

After you consider these factors, you can determine how many cloud controller cores you require. A typical 8 core, 8 GB of RAM server is sufficient for up to a rack of compute nodes — given the above caveats.

You must also consider key hardware specifications for the performance of user VMs. You must consider both budget and performance needs. Examples include: Storage performance (spindles/core), memory availability (RAM/core), network bandwidth (Gbps/core), and overall CPU performance (CPU/core).

For which metrics to track to determine how to scale your cloud, see the **Monitoring** chapter**.**

Adding Controller Nodes

You can facilitate the horizontal expansion of your cloud by adding nodes. Adding compute nodes is straightforward — they are easily picked up by the existing installation. However, you must consider some important points when you design your cluster to be highly available.

Recall that a cloud controller node runs several different services. You can install services that communicate only using the message queue internally — `nova-scheduler` and `nova-console` — on a new server for expansion. However, other integral parts require more care.

You should load balance user-facing services such as Dashboard, `nova-api` or the Object Storage proxy. Use any standard HTTP load balancing method (DNS round robin, hardware load balancer, software like Pound or HAProxy). One caveat with Dashboard is the VNC proxy, which uses the

WebSocket protocol — something that a L7 load balancer might struggle with. See also Horizon session storage (http://docs.openstack.org/developer/horizon/topics/deployment.html#session-storage).

You can configure some services, such as `nova-api` and `glance-api`, to use multiple processes by changing a flag in their configuration file — allowing them to share work between multiple cores on the one machine.

Several options are available for MySQL load balancing, and RabbitMQ has in-built clustering support. Information on how to configure these and many of the other services can be found in the **Operations Section.**

Segregating Your Cloud

Use one of the following OpenStack methods to segregate your cloud: *cells*, *regions*, *zones* and *host aggregates*. Each method provides different functionality, as described in the following table:

	Cells	Regions	Availability Zones	Host Aggregates
Use when	A single API endpoint is needed for compute, or a second level of scheduling is required	Discrete regions are needed, with separate API endpoints and no coordination between regions	You need to make a logical separation within your nova deployment for physical isolation or redundancy	A group of hosts has common features that you want to use to schedule.
Example	A cloud with multiple sites where VMs could be scheduled "anywhere", or a particular site.	A cloud with multiple sites, where VMs are always scheduled to a particular site and shared infrastructure is desired.	A single site cloud with equipment fed by separate power supplies.	Scheduling to hosts with trusted hardware support.
Overhead	A new service, `nova-cells`, must be run, each cell has a full nova installation except `nova-api`.	A different API endpoint for every region, each region has a full nova installation.	Configuration changes to nova.conf	Configuration changes to nova.conf
Shared services	Keystone, `nova-api`	Keystone	Keystone, all nova services	Keystone, All nova services

This array of options can be best divided into two — those which result in running separate nova deployments (cells & regions), and those which merely divide a single deployment (availability zones and host aggregates).

Cells and Regions

OpenStack Compute cells is designed to allow running the cloud in a distributed fashion without having to use more complicated technologies, or being invasive to existing nova installations. Hosts in a cloud are partitioned into groups called *cells*. Cells are configured in a tree. The top-level cell ("API cell") has a host that runs the `nova-api` service, but no `nova-compute` services. Each child cell runs all of the other typical `nova-*` services found in a regular installation, except for the `nova-api` service. Each cell has its own message queue and database service, and also runs `nova-cells` — which manages the communication between the API cell and child cells.

This allows for a single API server being used to control access to multiple cloud installations. Introducing a second level of scheduling (the cell selection), in addition to the regular `nova-scheduler` selection of hosts, provides greater flexibility to control where virtual machines are run.

Contrast this with regions. Regions have a separate API endpoint per installation, allowing for a more discrete separation. Users wishing to run instances across sites have to explicitly select a region. However, the additional complexity of a running a new service is not required.

The OpenStack Dashboard (Horizon) currently only uses a single region, so one dashboard service should be run per region. Regions are a robust way to share some infrastructure between OpenStack Compute installations, while allowing for a high degree of failure tolerance.

Availability Zones and Host Aggregates

Both availability zones and host aggregates partition a single nova deployment. While seeming similar to configure, host aggregates and availability zones differ in their intended use. The former allows the partition of OpenStack Compute deployments into logical groups for load balancing and instance distribution, the latter are used to provide some form of physical isolation and redundancy from other availability zones (e.g. by a using separate power supply or network equipment). Host aggregates can be regarded as a mechanism to further partitioning an availability zone, i.e. into multiple groups of hosts that share common resources like storage and network, or have a special property such as trusted computing hardware.

A common use of host aggregates is to provide information for use with the `nova-scheduler`. For example, limiting specific flavours or images to a subset of hosts.

Availability zones allow you to arrange sets of either OpenStack Compute or OpenStack Block Storage hosts into logical groups. You define the availability zone that a given Compute or block Storage host is in locally on each server. Availability zones are commonly used to identify a sets of servers that have some common attribute. For instance, if some of the racks in your data center are on a separate power source, you may put servers in those racks in their own availability zone.

Availability zones can also be helpful for separating out different classes of hardware. This is especially helpful with OpenStack Block Storage where you may have storage servers with different types of hard drives. When provisioning resources, users can specify what availability zone they would like their instance or volume to come from. This allows cloud consumers to ensure that their application resources are spread across multiple disparate machines in order to achieve high availability in the event of hardware failure.

Scalable Hardware

While several resources already exist to help with deploying and installing OpenStack, it's very important to make sure you have your deployment planned out ahead of time. This guide expects at least a rack has been set aside for the OpenStack cloud but also offers suggestions for when and what to scale.

Hardware Procurement

“The Cloud” has been described as a volatile environment where servers can be created and terminated at will. While this may be true, it does not mean that your servers must be volatile. Ensuring your cloud’s hardware is stable and configured correctly means your cloud environment will stay up and running. Basically, put effort into creating a stable hardware environment so you can host a cloud that users may treat as unstable and volatile.

OpenStack can be deployed on any hardware supported by an OpenStack-compatible Linux distribution, such as Ubuntu 12.04 as used in this books' reference architecture.

Hardware does not have to be consistent, but should at least have the same type of CPU in order to support instance migration.

The typical hardware recommended for use with OpenStack is "commodity". That is, very standard "value-for-money" offerings that most hardware vendors stock. It should be straightforward to divide your procurement into building blocks such as "compute", "object storage", and "cloud controller", and request as many of these as desired. Alternately should you be unable to to spend more, if you have existing servers, provided they meet your performance requirements and virtualisation technology, these are quite likely to be able to support OpenStack.

Capacity Planning

OpenStack is designed to increase in size in a straightforward manner. Taking into account the considerations in the **Scalability** chapter — particularly on the sizing of the cloud controller, it should be possible to procure additional compute or object storage nodes as needed. New nodes do not need to be the same specification, or even vendor, as existing nodes.

For compute nodes, `nova-scheduler` will take care of differences in sizing to do with core count and RAM amounts, however you should consider the user experience changes with differing CPU speeds. When adding object storage nodes, a weight should be specified that reflects the capability of the node.

Monitoring the resource usage and user growth will enable you to know when to procure. The **Monitoring** chapter details some useful metrics.

Burn-in Testing

Server hardware's chance of failure is high at the start and the end of its life. As a result, much effort in dealing with hardware failures while in production can be avoided by appropriate burn-in testing to attempt to trigger the early-stage failures. The general principle is to stress the hardware to its limits. Examples of burn-in tests include running a CPU or disk benchmark for several days.

Compute Nodes

Compute nodes form the resource core of the OpenStack Compute cloud, providing the processing, memory, network and storage resources to run instances.

CPU Choice

The type of CPU in your compute node is a very important choice. First, ensure the CPU supports virtualization by way of *VT-x* for Intel chips and *AMD-v* for AMD chips.

The number of cores that the CPU has also affects the decision. It's common for current CPUs to have up to 12 cores. Additionally, if the CPU supports Hyper-threading, those 12 cores are doubled to 24 cores. If you purchase a server that supports multiple CPUs, the number of cores is further multiplied.

Whether you should enable hyper-threading on your CPUs will depend upon your use case. We recommend you do performance testing with your local workload with both hyper-threading on and off to determine what is more appropriate in your case.

Hypervisor choice

OpenStack Compute supports many hypervisors to various degrees, including KVM, LXC, QEMU, UML, VMWare ESX/ESXi, Xen, PowerVM, Hyper-V.

Probably the most important factor in your choice of hypervisor is your current usage or experience. Aside from that, there are practical concerns to do with feature parity, documentation, and the level of community experience. For example, KVM is the most widely adopted hypervisor in the OpenStack community. Besides KVM, more deployments exist running Xen, LXC, VMWare and Hyper-V than the others listed — however, each of these are lacking some feature support or the documentation on how to use them with OpenStack is out of date.

The best information available to support your choice is found on the Hypervisor Support Matrix (https://wiki.openstack.org/wiki/HypervisorSupportMatrix), and in the reference manual (http://docs.openstack.org/folsom/openstack-compute/admin/content/ch_hypervisors.html).

Note: It is also possible to run multiple hypervisors in a single deployment using Host Aggregates or Cells. However, an individual compute node can only run a single hypervisor at a time.

Instance Storage Solutions

As part of the procurement for a compute cluster, you will need to specify some storage for the disk on which the instantiated instance runs. There are three main approaches to providing this temporary-style storage, and it is important to understand the implications of the choice. They are:

Off compute node storage – shared file system
On compute node storage – shared file system
On compute node storage – non-shared file system

In general, the questions you should be asking when selecting the storage are as follows:

What is the platter count you can achieve?
Do more spindles result in better I/O despite network access?
Which one results in the best cost-performance scenario you're aiming for?
How do you manage the storage operationally?

Off Compute Node Storage – Shared File System

Many operators use separate compute and storage hosts. Compute services and storage services have different requirements, compute hosts typically require more CPU and RAM than storage hosts. Therefore, for a fixed budget, it makes sense to have different configurations for your compute nodes and your storage nodes with compute nodes invested in CPU and RAM, and storage nodes invested in block storage.

Also, if you use separate compute and storage hosts then you can treat your compute hosts as "stateless". This simplifies maintenance for the compute hosts. As long as you don't have any instances currently running on a compute host, you can take it offline or wipe it completely without having any effect on the rest of your cloud. However, if you are more restricted in the number of physical hosts you have available for creating your cloud and you want to be able to dedicate as many of your hosts as possible to running instances, it makes sense to run compute and storage on the same machines.

In this option, the disks storing the running instances are hosted in servers outside of the compute nodes. There are also several advantages to this approach:

If a compute node fails, instances are usually easily recoverable.
Running a dedicated storage system can be operationally simpler.
Being able to scale to any number of spindles.
It may be possible to share the external storage for other purposes.

The two main downsides to this approach are:

Depending on design, heavy I/O usage from some instances can affect unrelated instances.
Use of the network can decrease performance.

On Compute Node Storage – Shared File System

In this option, each `nova-compute` node is specified with a significant amount of disks, but a distributed file system ties the disks from each compute node into a single mount. The main advantage of this approach is:

If additional storage is required, this option can scale to external storage.

This approach does have several downsides:

Running a distributed file system can make you lose your data locality compared with non-shared storage.
Recovery of instances is complicated by depending on multiple hosts.
The chassis size of the compute node can limit the number of spindles able to be used in a compute node.
Use of the network can decrease performance.

On Compute Node Storage – Non-shared File System

In this option, each `nova-compute` node is specified with enough disks to store the instances it hosts. There are two main reasons why this is a good idea:

Heavy I/O usage on one compute node does not affect instances on other compute nodes.
Direct I/O access can increase performance.

This has several downsides:

If a compute node fails, the instances running on that node will be lost.
The chassis size of the compute node can limit the number of spindles able to be used in a compute node.
Migrations of instances from one node to another are more complicated, and rely on features which may not continue to be developed.
If additional storage is required, this option does not to scale.

Issues with Live Migration

We consider live migration an integral part of the operations of the cloud. This feature provides the ability to seamlessly move instances from one physical host to another, a necessity for performing upgrades that require reboots of the compute hosts, but only works well with shared storage. Theoretically live migration can be done with non-shared storage, using a feature known as *KVM live block migration*. However, this is a little-known feature in OpenStack, with limited testing when compared to live migration, and is slated for deprecation in KVM upstream.

Choice of File System

If you want to support shared storage live migration, you'll need to configure a distributed file system. Possible options include:

NFS (default for Linux)
GlusterFS
MooseFS
Lustre

We've seen deployments with all, and recommend you choose the one you are most familiar with operating.

Overcommitting

OpenStack allows you to overcommit CPU and RAM on compute nodes. This allows you to increase the number of instances you can have running on your cloud, at the cost of reducing the performance of the instances. OpenStack Compute uses the following ratios by default:

CPU allocation ratio: 16
RAM allocation ratio: 1.5

The default CPU allocation ratio of 16 means that the scheduler will allocate up to 16 virtual cores on a node per physical cores. For example, if a physical node has 12 cores, and each virtual machine instance uses 4 virtual cores, the scheduler will allocate up to 192 virtual cores to instances (e.g., 48 instances, in the case where each instance has 4 virtual cores).

Similarly, the default RAM allocation ratio of 1.5 means that the scheduler will allocate instances instances to a physical node as long as the total amount of RAM associated with the instances is less than 1.5 times the amount of RAM available on the physical node. For example, if a physical

node has 48 GB of RAM, the scheduler will allocate instances to that node until the sum of the RAM associated with the instances reaches 72 GB (e.g. 6 instances, in the case where each instance has 8GB of RAM).

You will need to select the appropriate CPU and RAM allocation ratio for your particular use case.

Logging

Logging is detailed more fully in the **Logging and Monitoring** chapter, however it is an important design consideration to take into account before commencing operations of your cloud. OpenStack produces a great deal of useful logging information, however, in order for it to be useful for operations purposes you should consider having a central logging server to send logs to, and a log parsing/analysis system (e.g. logstash).

Networking

Networking in OpenStack is a complex, multi-faceted challenge. As such, it is covered in the **Network Design** chapter.

Decisions for Storage

Storage is found in many parts of the OpenStack stack, and the differing types can cause confusion to even experienced cloud engineers. This section focuses on persistent storage options you may configure with your cloud.

OpenStack Storage Concepts

OpenStack Storage

	Ephemeral storage	**Block storage**	**Object storage**
Used for...	running operating system and scratch space	adding additional persistent storage to a virtual machine (VM)	storing data, including VM images
Accessed via...	A file system	a block device which can be partitioned, formatted and mounted (e.g., /dev/vdc)	REST API
Accessible from...	within a VM	within a VM	anywhere
Managed by...	OpenStack Compute (Nova)	OpenStack Block Storage (Cinder)	OpenStack Object Storage (Swift)
Persists until...	VM is terminated	deleted by user	deleted by user
Sizing determined by...	administrator configures size settings, known as *flavors*	specified by user in initial request	amount of available physical storage
Example of typical usage:	10GB first disk, 30GB second disk	1TB disk	10s of TBs of dataset storage

If you only deploy the OpenStack Compute Service (nova), your users will not have access to any form of persistent storage by default. The disks associated with virtual machines are "ephemeral", meaning that (from the user's point of view) they effectively disappear when a virtual machine is

terminated. You need to identify what type of persistent storage you want to support for your users.

Today, OpenStack clouds explicitly support two types of persistent storage: *object storage* and *block storage*.

Object storage

With object storage, users access binary objects through a REST API. You may be familiar with Amazon S3, which is a well-known example of an object storage system. If your intended users need to archive or manage large datasets, you will likely want to provide them with object storage. In addition, OpenStack can store your virtual machine images inside of an object storage system, as an alternative to storing the images on a file system.

Block storage

Block storage (sometimes referred to as volume storage) exposes a block device to the user. Users interact with block storage by attaching volumes to their running virtual machine instances. These volumes are persistent: they can be detached from one instance and re-attached to another, and the data will remain intact. Block storage is implemented in OpenStack by the OpenStack Block Storage (Cinder) project, which supports multiple backends in the form of drivers. Your choice of a storage backend must be supported by a Block Storage driver.

File-level storage

With file-level storage, users access stored data using the operating system's file system interface. Most users, if they have used a network storage solution before, have encountered this form of networked storage. In the Unix world, the most common form of this is NFS. In the Windows world, the most common form is called CIFS (previously, SMB). OpenStack clouds do not present file-level storage to end users. However, it is important to consider file-level storage for storing instances under `/var/lib/nova/instances` when designing your cloud, since you must have a shared file system if you wish to support live migration.

Choosing Storage Backends

In general, the questions you should be asking when selecting the storage are:

Do my users need block storage?
Do my users need object storage?
Do I need to support live migration?

Should my persistent storage drives be contained in my compute nodes, or should I use external storage?
What is the platter count I can achieve? Do more spindles result in better I/O despite network access?
Which one results in the best cost-performance scenario I'm aiming for?
How do I manage the storage operationally?
How redundant and distributed is the storage? What happens if a storage node fails? To what extent can it mitigate my data-loss disaster scenarios?

If you wish to deploy your storage using entirely commodity hardware, there are a number of open-source packages that you can use, as shown in the table below.

	Object	Block	File-level* (live migration support)
Swift	✓		
LVM		✓	
Ceph	✓	✓	experimental
Gluster	✓		✓
ZFS		✓	
Sheepdog		experimental	

* This list of open-source file-level shared storage solutions is not exhaustive, other open source solutions exist (e.g., NFS, MooseFS). Your organization may already have deployed a file-level shared storage solution which you can use.

In addition to the open-source technologies, there are a number of proprietary solutions that are officially supported by OpenStack Block Storage. They are offered by the following vendors:

IBM (Storwize family/SVC, XIV)
NetApp
Nexenta
SolidFire

You can find a matrix of the functionality provided by all of the supported Block Storage drivers on the OpenStack wiki (https://wiki.openstack.org/wiki/CinderSupportMatrix).

Also, you need to decide whether you want to support object storage in your cloud. The two common use cases for providing object storage in a compute cloud are:

To provide users with a persistent storage mechanism
As a scalable, reliable data store for virtual machine images

Commodity Storage Backend Technologies

Here we provide a high-level overview of the differences among the different commodity storage backend technologies.

OpenStack Object Storage (Swift) is the official OpenStack Object Store implementation. It is a mature technology that has been used for several years in production by Rackspace as the technology behind Rackspace Cloud Files. As it is highly scalable, it is well-suited to managing petabytes of storage. OpenStack Object Storage's advantages are better integration with OpenStack (integrates with OpenStack Identity, works with OpenStack Dashboard interface), and better support for multiple data center deployment through support of asynchronous eventual consistency replication. Therefore, if you eventually plan on distributing your storage cluster across multiple data centers, if you need unified accounts for your users for both compute and object storage, or if you want to control your object storage with the OpenStack dashboard, you should consider OpenStack Object Storage. More detail can be found about OpenStack Object Storage in the section below.

Ceph is a scalable storage solution that replicates data across commodity storage nodes. Ceph was originally developed by one of the founders of DreamHost and is currently used in production there. Ceph was designed to expose different types of storage interfaces to the end-user: it supports object storage, block storage, and file system interfaces, although the file system interface is not yet considered production-ready. Ceph supports the same API as Swift for object storage, can be used as a back-end for Cipher block storage, as well as back-end storage for Glance images. Ceph supports "thin provisioning", implemented using copy-on-write. This can be useful when booting from volume because a new volume can be provisioned very quickly. However, Ceph does not support keystone-based authentication, so if you want to provide users with access to object storage, you will need to create new accounts for them with your Ceph deployment in addition to the accounts they have with your OpenStack cloud. Ceph's advantages are that it gives the administrator more fine-grained control over data distribution and replication strategies, enables you to consolidate your object and block storage, enables very fast provisioning of boot-from-volume instances using thin provisioning, and supports a distributed file system interface,

though this interface is not yet recommended (http://ceph.com/docs/master/faq/) for use in production deployment by the Ceph project. If you wish to manage your object and block storage within a single system, or if you wish to support fast boot-from-volume, you should consider Ceph.

Gluster is a distributed, shared file system. As of Gluster version 3.3, you can use Gluster to consolidate your object storage and file storage into one unified file and object storage solution, which is called Gluster UFO. Gluster UFO uses a customizes version of Swift that uses Gluster as the back-end. The main advantage of using Gluster UFO over regular Swift is if you also want to support a distributed file system, either to support shared storage live migration or to provide it as a separate service to your end-users. If you wish to manage your object and file storage within a single system, you should consider Gluster UFO.

LVM refers to Logical Volume Manager, a Linux-based system that provides an abstraction layer on top of physical disks to expose logical volumes to the operating system. The LVM (Logical Volume Manager) backend implements block storage as LVM logical partitions. On each host that will house block storage, an administrator must initially create a volume group dedicated to Block Storage volumes. Blocks will be created from LVM logical volumes. Note that LVM does *not* provide any replication. Typically, administrators configure RAID on nodes that use LVM as block storage, in order to protect against failures of individual hard drives. However, RAID does not protect against a failure of the entire host.

The Solaris iSCSI driver for OpenStack Block Storage implements blocks as **ZFS** entities. ZFS is a file system that also has functionality of a volume manager. This is unlike on a Linux system, where there is a separation of volume manager (LVM) and file system (e.g., ext3, ext4, xfs, btrfs). ZFS has a number of advantages over ext4, including improved data integrity checking. The ZFS backend for OpenStack Block Storage only supports Solaris-based systems such as Illumos. While there is a Linux port of ZFS, it is not included in any of the standard Linux distributions, and it has not been tested with OpenStack Block Storage. As with LVM, ZFS does not provide replication across hosts on its own, you need to add a replication solution on top of ZFS if your cloud needs to be able to handle storage node failures. We don't recommend ZFS unless you have previous experience with deploying it, since the ZFS backend for Block Storage requires a Solaris-based operating system and we assume that your experience is primarily with Linux-based systems.

Sheepdog is a recent project that aims to provide block storage for KVM-based instances, with support for replication across hosts. We don't recommend Sheepdog for a production cloud, because its authors at NTT Labs consider Sheepdog as an experimental technology.

Notes on OpenStack Object Storage

OpenStack Object Storage provides a highly scalable, highly available storage solution by relaxing some of the constraints of traditional file systems. In designing and procuring for such a cluster, it is important to understand some key concepts about its operation. Essentially, this type of storage is built on the idea that all storage hardware will fail, at every level, at some point. Infrequently encountered failures that would hamstring other storage systems, such as issues taking down RAID cards, or entire servers are handled gracefully with OpenStack Object Storage.

A good document describing the Object Storage architecture is found within the developer documentation (http://docs.openstack.org/developer/swift/overview_architecture.html) - read this first. Once you have understood the architecture, you should know what a proxy server does and how zones work. However, some there important points that are often missed at first glance.

When designing your cluster, you must consider durability and availability. Understand that the predominant source of these is the spread and placement of your data, rather than the reliability of the hardware. Consider the default value of the number of replicas, which is 3. This means that when before an object is marked as having being written at least two copies will exist - in case a single server fails to write, the third copy may or may not yet exist when the write operation initially returns. Altering this number increases the robustness of your data, but reduces the amount of storage you have available. Next look at the placement of your servers. Consider spreading them widely throughout your data centre's network and power failure zones. Is a zone a rack, a server or a disk?

Object Storage's network patterns may be unfamiliar at first. There are three main traffic flows to consider: among object/container/account servers, between those servers and the proxies, and between the proxies and your users. Object Storage is very 'chatty' among servers hosting data - even a small cluster will do megabytes/second of traffic which is predominantly "Do you have the object?"/"Yes I have the object!". Of course, if the answer to the aforementioned question is negative or times out, then replication of the object begins. Consider the scenario where an entire server fails, and 24TB of data needs to be transferred "immediately" to remain at 3 copies - this can put significant load on the network.

Another oft forgotten fact is that when a new file is being uploaded, the proxy server must write out as many streams as there are replicas - giving a multiple of network traffic. For a 3-replica cluster, 10Gbps in means 30Gbps out. Combining this with the previous high bandwidth demands of replication is what results in the recommendation that your private network is of significantly higher bandwidth than your public need be. Oh, and OpenStack Object Storage communicates internally with unencrypted, unauthenticated rsync for performance - you do want the private network to be private.

The remaining point on bandwidth is the public facing portion. swift-proxy is stateless, meaning you can easily add more and use http load-balancing methods to share bandwidth and availability between them. More proxies means more bandwidth, if your storage can keep up.

Network Design

OpenStack provides a rich networking environment, and this chapter details the requirements and options to deliberate when designing your cloud.

If this is the first time you are deploying a cloud infrastructure in your organisation, after reading this section, your first conversations should be with your networking team. Network usage in a running cloud is vastly different from traditional network deployments, and has the potential to be disruptive at both a connectivity and a policy level.

For example, the number of IP addresses that you will need for both your guest instances as well as management infrastructure will need proper planning. Additionally, cloud network connectivity through proxy servers and firewalls also needs researched and discussed.

Management Network

A management network, typically consisting of a separate switch and separate NICs, is a recommended option. This allows system administration and monitoring system access from being disrupted.

Consider creating other private networks for communication between internal components of OpenStack, such as the Message Queue and OpenStack Compute. VLANs are great for these scenarios.

Public Addressing Options

There are two main types of IP addresses for guest virtual machines: Fixed IPs and Floating IPs. Fixed IPs are assigned to instances on boot, whereas Floating IP addresses can change their association between instances by action of the user. Both types of IP addresses can either be public or private, depending on your use case.

Fixed IP addresses are required, whereas it is possible to run OpenStack without Floating IPs. One of the most common use cases for Floating IPs is to provide public IP addresses to a private cloud, where there are a limited number of IP addresses available. Another is for a public cloud user to have a "static" IP address that can be reassigned when an instance is upgraded or moved.

Fixed IP addresses can be private for private clouds, or public for public clouds. When an instance terminates, its Fixed IP is lost. It is worth noting that newer users of cloud computing may find their ephemeral nature frustrating.

IP Address planning

An OpenStack installation can potentially have many subnets, and different types of services in each. An IP address plan can assist with a shared understanding of network partition purposes and scalability. Control services can have public and private IP addresses, and as noted above there are a couple of options for instance's public addresses.

An IP address plan might be broken down into the following sections:

subnet router	Packets leaving the subnet go via this address, which could be a dedicated router or a nova-network service.
control services public interfaces	Public access to `swift-proxy`, `nova-api`, `glance-api` and horizon come to these addresses, which could be on one side of a load balancer, or pointing at individual machines.
Object Storage cluster internal communications	Traffic amongst object/account/container servers and between these and the proxy server's internal interface uses this private network.
compute and storage communications	If ephemeral or block storage storage is external to the compute node, this network will be used.
out-of-band remote management	If a dedicated remote access controller chip is included in servers, often these are on a separate network.
in-band remote management	Often, an extra (eg 1G) interface on compute or storage nodes is used for system administrators or monitoring tools to access the host instead of going via the public interface.
spare space for future growth	Adding more public-facing control services, or guest instance IPs should always be part of your plan.

For example, take a deployment which has both OpenStack Compute and Object Storage, with private ranges 172.22.42.0/24 and 172.22.87.0/26 available. One way to segregate the space might be:

```
172.22.42.0/24
172.22.42.1   - 172.22.42.3   - subnet routers
172.22.42.4   - 172.22.42.20  - spare for networks
172.22.42.21  - 172.22.42.104 - Compute node remote access controllers (inc spare)
172.22.42.105 - 172.22.42.188 - Compute node management interfaces (inc spare)
172.22.42.189 - 172.22.42.208 - Swift proxy remote access controllers (inc spare)
172.22.42.209 - 172.22.42.228 - Swift proxy management interfaces (inc spare)
172.22.42.229 - 172.22.42.252 - Swift storage servers remote access controllers (inc spare)
172.22.42.253 - 172.22.42.254 - spare
172.22.87.0/26:
172.22.87.1  - 172.22.87.3    - subnet routers
172.22.87.4  - 172.22.87.24   - Swift proxy server internal interfaces (inc spare)
172.22.87.25 - 172.22.87.63   - Swift object server internal interfaces (inc spare)
```

A similar approach can be taken with public IP addresses, taking note that large, flat ranges are preferred for use with guest instance IPs. Take into account that for some OpenStack networking options, a public IP address in the range of a guest instance public IP address is assigned to the nova-compute host.

Network Topology

OpenStack Compute provides several network managers, each with their own strengths and weaknesses. The selection of a network manager changes your network topology, so the choice should be made carefully.

Type	Strengths	Weaknesses
Flat	Extremely simple. No DHCP broadcasts.	Requires file injection into the instance. Limited to certain distributions of Linux. Difficult to configure and is not recommended.
FlatDHCP	Relatively simple to setup. Standard networking. Works with all operating systems.	Requires its own DHCP broadcast domain.
VlanManager	Each tenant is isolated to their own VLANs.	More complex to set up. Requires its own DHCP broadcast domain. Requires many VLANs to be trunked onto a single port. Standard VLAN number limitation. Switches must support 802.1q VLAN tagging.
FlatDHCP Multi-host HA	Networking failure is isolated to the VMs running on the hypervisor affected. DHCP traffic can be isolated within an individual host. Network traffic is distributed to the compute nodes.	More complex to set up. By default, compute nodes need public IP addresses. Options must be carefully configured for live migration to work with networking.

VLANs

VLAN configuration can be as simple or as complicated as desired. The use of VLANs has the benefit of allowing each project its own subnet and broadcast segregation from other projects. In order to allow OpenStack to efficiently use VLANs, you will need to allocate a VLAN range (one for each project) and turn each compute node switch port into a trunk port.

For example, if you estimate that your cloud will support a max of 100 projects, pick a free VLAN range that your network infrastructure is currently not using (e.g. VLAN 200 - 299). You will then need to configure OpenStack with this range as well as configure your switch ports to allow VLAN traffic from that range.

Multi-NIC

OpenStack Compute has the ability to assign multiple NICs to instances on a per-project basis. This is generally an advanced feature and not an everyday request. This can easily be done on a per-request basis, though. However, be aware that a second NIC will use up an entire subnet or VLAN. This will decrement your total number of supported projects by one.

Multi-host and Single-host Networking

The nova-network service has the ability to operate in a multi-host or single-host mode. Multi-host is when each compute node runs a copy of nova-network and the instances on that compute node will use the compute node as a gateway to the Internet. The compute nodes also host the Floating IPs and Security Groups for instances on that node. Single-host is when a central server, for example, the cloud controller, runs the `nova-network` service. All compute nodes forward traffic from the instances to the the cloud controller. The cloud controller then forwards traffic to the Internet. The cloud controller hosts the Floating IPs and Security Groups for all instances on all compute nodes in the cloud.

There are benefits to both modes. Single-node has the downside of a single point of failure. If the cloud controller is not available, instances will not be able to communicate on the network. This is not true with multi-host, but multi-host requires that each compute node has a public IP address in order to communicate on the Internet. If you are not able to obtain a significant block of public IP addresses, multi-host might not be an option.

Services for Networking

OpenStack, like any network application, has a number of the standard considerations to apply, such as DNS and NTP.

NTP

Time synchronisation is a critical element to ensure continued operation of OpenStack components. Correct time is necessary to avoid errors in instance scheduling, replication of objects in the object store, and even matching log timestamps for debugging.

All servers running OpenStack components should be able to access an appropriate NTP server. You may decide to set one up locally, or use the public pools available from http://www.pool.ntp.org/

DNS

OpenStack does not currently provide DNS services, aside from the dnsmasq daemon which resides on `nova-network` hosts. You could consider providing a dynamic DNS service to allow instances to update a DNS entry with new IP addresses. You can also consider making a generic forward and reverse DNS mapping for instance's IP addresses, such as vm-203-0-113-123.example.com.

Example Architecture

Since OpenStack is highly configurable, with many different backends and network configuration options, it is difficult to write documentation that covers all possible OpenStack deployments. Therefore, in this guide we have defined an *example architecture* to simplify the task of documenting, as well as to scope this guide so that it is focused on a configuration where the authors have direct deployment experience.

Overview

OpenStack release	Folsom
Host operating system	Ubuntu 12.04 LTS
OpenStack package repository	Ubuntu Cloud Archive (https://wiki.ubuntu.com/ServerTeam/CloudArchive) *
Hypervisor	KVM
Database	MySQL*
Message queue	RabbitMQ
Networking service	nova-network
Network manager	FlatDHCP
Single nova-network or multi-host?	multi-host*
Image Service (glance) backend	file
Identity Service (keystone) driver	SQL
Block Storage Service (cinder) backend	LVM/iSCSI
Live Migration backend	shared storage using NFS *
Object storage	OpenStack Object Storage (swift)

An asterisk (*) indicates when the example architecture deviates from the settings of a default installation.

Note the following features of OpenStack are supported by the example architecture documented in this guide, but are optional:

Dashboard
Block storage
Floating IPs
Live migration
Object storage

Rationale

This example architecture has been selected based on the current default feature set of OpenStack Folsom, with an emphasis on stability. In particular, if none of the guide authors had experience deploying the Folsom release of OpenStack with a specific backend or configuration, we did not consider it for the example architecture. We believe that many clouds currently running OpenStack in production will have made similar choices.

The first selection to make is the Operating System that will run on all of the physical nodes. While OpenStack is supported on several distributions of Linux, we used **Ubuntu 12.04 LTS (Long Term Support)**. This is due to several factors, including its use by the majority of the development community, feature completeness compared with other distributions and clear future support plans.

We recommend that you do not use the default Ubuntu OpenStack install packages and instead use the Ubuntu Cloud Archive (https://wiki.ubuntu.com/ServerTeam/CloudArchive). The Cloud Archive is a package repository supported by Canonical that will allow you to upgrade to future OpenStack releases while still remaining on Ubuntu 12.04.

KVM as a hypervisor complements the choice of Ubuntu - being a matched pair in terms of support, and also because of the significant degree of attention it garners from the OpenStack development community (including the authors, who mostly use KVM). It is also feature complete, free from licensing charges and restrictions.

MySQL follows a similar trend. Despite its recent change of ownership, this database is the most tested for use with OpenStack and is heavily documented for running on Ubuntu. We deviate from the default database, *SQLite*, because SQLite is not an appropriate database for production usage.

The choice of **RabbitMQ** over other AMQP compatible options that are gaining support in OpenStack, such as ZeroMQ and Qpid is due to its ease of use with Ubuntu and significant testing in production. It also is the only option which supports features such as Compute Cells. We recommend clustering with RabbitMQ, as it is an integral component of the system, and fairly simple to implement due to its inbuilt nature.

As discussed in previous chapters, there are several options for networking in OpenStack Compute. We recommend **FlatDHCP** and to use **Multi-Host** networking mode for high availability, running one `nova-network` daemon per OpenStack Compute host. This provides a robust mechanism for ensuring network interruptions are isolated to individual compute hosts, and allows for the direct use of hardware network gateways.

Live Migration is supported by way of shared storage, with **NFS** as the distributed file system.

Acknowledging that many small-scale deployments see running an Object Storage service just for the storage of virtual machine images as too costly, we opted for the file backend in the OpenStack Image Catalogue and Delivery Service (Glance). If the cloud you are designing also intends to run Object Storage, it is trivial to enable this as the backend instead, and a recommended approach.

The **SQL backend for Identify Service (keystone)** was selected over others, such as LDAP, as it is the simplest to install and has a large degree of robustness. The authors acknowledge that many installations will desire binding with an existing directory services, and caution careful understanding of the array of options available (http://docs.openstack.org/folsom/openstack-compute/admin/content/reference-for-ldap-config-options.html)

The Block Storage service (Cinder) will be installed natively on external storage nodes and will use the **LVM/iSCSI plugin**. Most Block Storage Service plugins are tied to particular vendor products and implementations limiting their use to consumers of those hardware platforms, but LVM/iSCSI is known to be robust and stable on commodity hardware.

While the cloud can be run without the **OpenStack Dashboard**, we consider it to be indispensable, not just for user interaction with the cloud, but also as a tool for operators. Additionally, the dashboard's use of Django makes it a flexible framework for extension.

Why not the OpenStack Network Service (quantum)?

We do not discuss the OpenStack Network Service (quantum) in this guide, because the authors of this guide only have production deployment experience using `nova-network`. Additionally, it does not yet support multi-host networking.

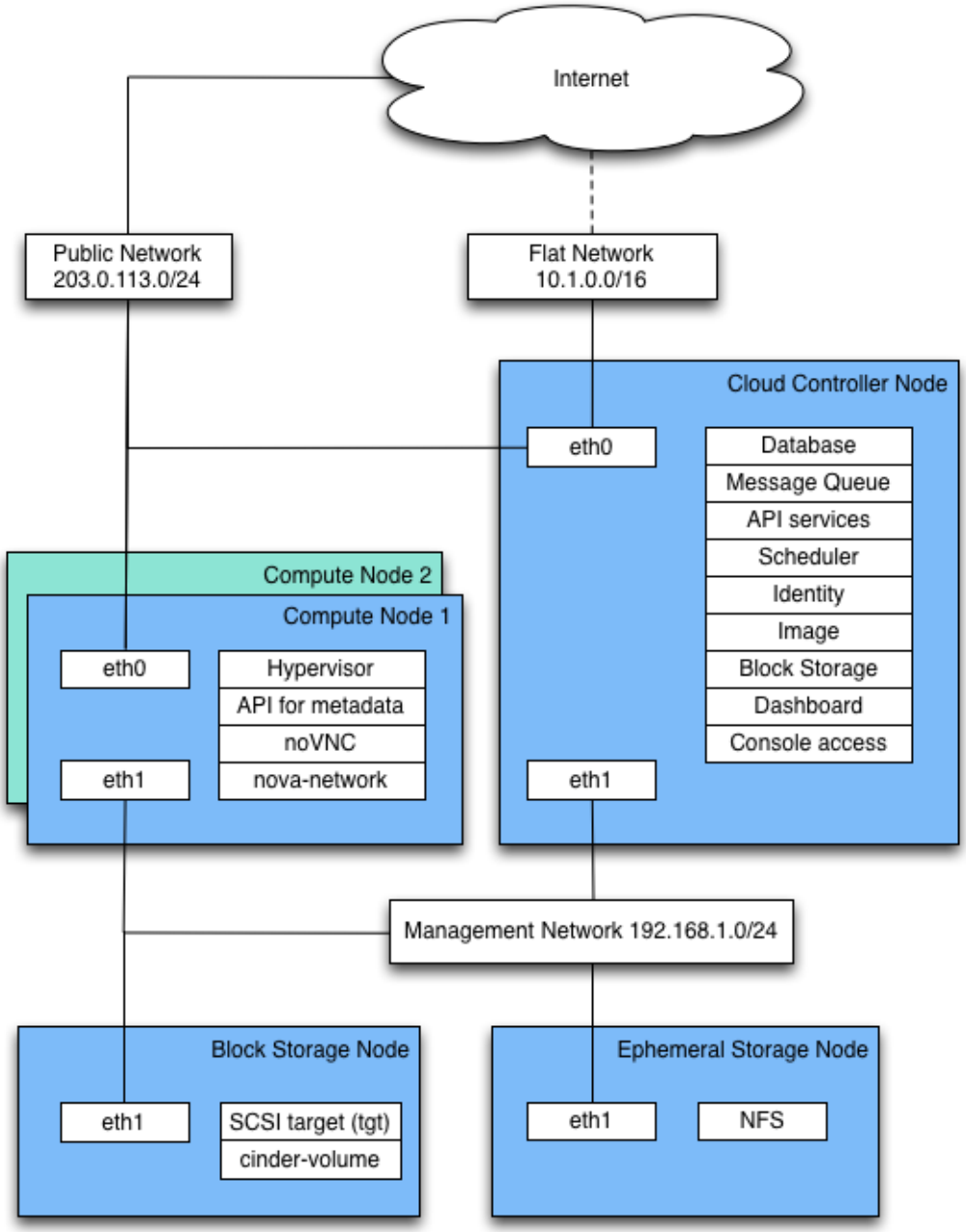
Why multi-host Networking?

In a default OpenStack deployment, there is a single `nova-network` service that runs within the cloud (usually on the cloud controller) that provides services such as network address translation (NAT), DHCP, and DNS to the guest instances. If the single node that runs the `nova-network` service goes down, then you will not be able to access your instances nor will the instances be able to access the Internet. There is also the risk that the single node that runs the nova-network service can become a bottleneck if there is excessive network traffic coming in and out of the cloud.

Multi-host (http://docs.openstack.org/folsom/openstack-compute/admin/content/existing-ha-networking-options.html#d6e8906) is a high-availability option for the network configuration where the nova-network service is run on every compute node instead of running on only a single node.

Detailed Description

The reference architecture consists of multiple compute nodes, a cloud controller, an external NFS storage server for instance storage and an OpenStack Block Storage server for volume storage. A network time service (Network Time Protocol, NTP) synchronizes time for all the nodes. FlatDHCPManager in multi-host mode is used for the networking.

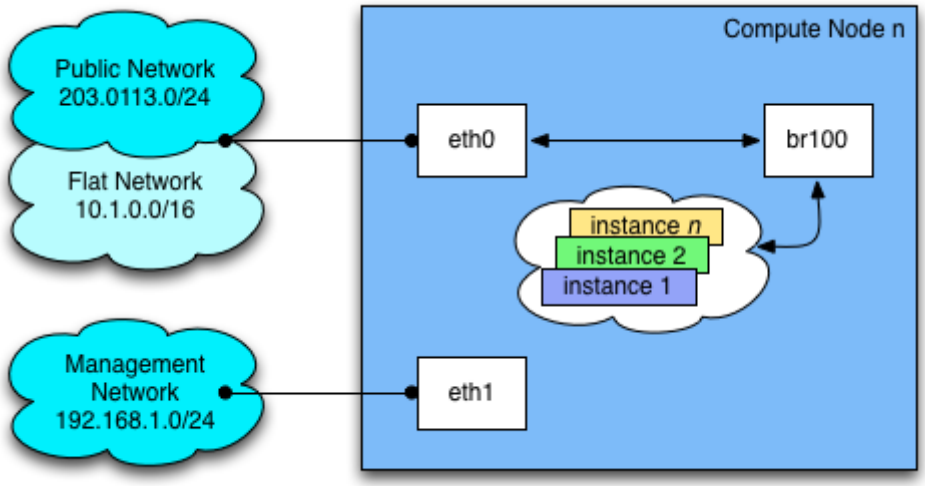


The cloud controller runs: the Dashboard, the API services, the database (MySQL), a message queue server (RabbitMQ), the scheduler for choosing compute resources (nova-scheduler), Identity services (keystone, `nova-consoleauth`), Image services (`glance-api`, `glance-registry`),

services for console access of guests, and block storage services including the scheduler for storage resources (`cinder-api` and `cinder-scheduler`).

Compute nodes are where the computing resources are held, and in our example architecture they run the hypervisor (KVM), libvirt (the driver for the hypervisor which enables live migration node to node), `nova-compute`, `nova-api-metadata` (generally only used when running in multi-host mode, it retrieves instance-specific metadata), nova-vncproxy, and `nova-network`.

The network consists of two switches, one for the management or private traffic, and one which covers public access including Floating IPs. To support this, the cloud controller and the compute nodes have two network cards. The OpenStack Block Storage and NFS storage servers only need to access the private network and therefore only need one network card, but multiple cards run in a bonded configuration are recommended if possible. Floating IP access is direct to the internet, whereas Flat IP access goes via a NAT.



Optional Extensions

There are several areas where this reference architecture could be extended:

Adding additional cloud controllers (see **Maintenance, Debugging and Failures**)
Adding an OpenStack Storage service (http://docs.openstack.org/folsom/openstack-object-storage/admin/)
Adding additional OpenStack Block Storage hosts (see **Maintenance, Debugging and Failures**)

What Next

Congratulations! By this point you should have a solid design of your cloud. We now recommend you turn to the OpenStack Install and Deploy Manual - Ubuntu (http://docs.openstack.org/folsom/openstack-compute/install/apt/content/), which contains a step-by-step guide on how to manually install the OpenStack packages and dependencies on your cloud. While it is important for an operator to be familiar with the steps involved in deploying OpenStack, we also strongly encourage you to evaluate configuration management tools such as Puppet or Chef that can help automate this deployment process.

In the remainder of the guide, we will assume that you have successfully deployed an OpenStack cloud and are able to perform basic operations such as adding images, booting instances, and attaching volumes.

As your focus turns to stable operations, we recommend you do an initial skim of the remainder of the book to get a sense of the content. Some of this content will be useful to read in advance, so that you can put best practices into effect that will simplify your life in the long run. Other content is more useful as a reference that you may only need to refer to in the case of an unexpected event (such a power failure or troubleshooting a particular problem).

Operations

- Lay of the Land
- Managing Projects and Users
- User Facing Operations
- Maintenance, Failures, and Debugging
- Network Troubleshooting
- Logging and Monitoring
- Backups and Recovery
- Customize
- Upstream OpenStack

Lay of the Land

From this point forward in the guide, we assume that you have an OpenStack cloud up and running. This section will assist in setting up your working environment and use it to take a look around your cloud.

Client Command Line Tools

We recommend using a combination of the OpenStack command line interface (CLI) client tools and the OpenStack Dashboard. Some users with a background in other cloud technologies may be using the EC2 Compatibility API, which uses somewhat different naming conventions from the native API which we will highlight where required.

We strongly suggest you obtain the command line clients from the Python Package Index(PyPI) (https://pypi.python.org/) instead of from the Ubuntu or Fedora packages. The clients are under heavy development and it is very likely at any given time the version of the packages distributed by your operating system vendor are out of date. The "pip" utility is used to manage package installation from the PyPI archive and is available in the "python-pip" package in most Linux distributions. Each OpenStack project has its own client, so depending on which services your site runs you will needs some or all of the following packages:

python-novaclient (nova CLI)
python-glanceclient (glance CLI)
python-keystoneclient (keystone CLI)
python-cinderclient (cinder CLI)
python-swiftclient (swift CLI)
python-quantumclient (quantum CLI)

Installing the Tools

To install (or upgrade) a package from the PyPI archive with pip, as root:

```
# pip install [--upgrade] <package-name>
```

To remove the package:

```
# pip uninstall <package-name>
```

If you need even newer versions of the clients, pip can install directly from the upstream git repository using the `-e` flag. You will also need to specify a name for the Python egg that will be installed. For example:

```
# pip install -e git+https://github.com/openstack/python-novaclient.git#egg=python-novaclient
```

If you support the EC2 API on your cloud you should also install the "euca2ools" package or some other EC2 API tool so you can get the same view your users have. Using EC2 API based tools is mostly out of the scope of this guide, though we will discuss getting credentials for use with it.

Administrative Command Line Tools

There are also several *-manage command line tools:

nova-manage
glance-manage
keystone-manage
cinder-manage

Unlike the tools mentioned above, the `*-manage` tools must be run from the cloud controller, as root, because they need read access to the config files such as `/etc/nova/nova.conf` and will make queries directly against the database rather than against the OpenStack API endpoints.

The existence of the `*-manage` tools is a legacy issue. It is a goal of the OpenStack project to eventually migrate all of the remaining functionality in the `*-manage` tools into the regular client tools. Until that day, you will still need to SSH into the cloud controller node to perform some maintenance operations that require one of the `*-manage` tools.

Getting Credentials

You must have the appropriate credentials if you wish to use the command line tools to make queries against your OpenStack cloud. By far the easiest way to obtain authentication credentials to use with command line clients is to use the Horizon dashboard. From the top right navigation row select the "Settings" link. This will bring you to the user settings page where you can set your language and timezone preferences for the dashboard view, but more importantly it will also change the left hand navigation column to include "OpenStack API" and "EC2 Credentials" links. These links allow you to generate files you can source in your shell to populate the environment variables the command line tools need to know where your service endpoints are as well as your authentication information.

For using the OpenStack native tools follow the "OpenStack API" link. The top section will list the URLs of your service endpoints and below that is a section titled "Download OpenStack RC File". For looking at the cloud as an administrator, you can choose admin from the drop-down menu. In this section select the project you wish to get credentials for and click the "Download RC" button. This will generate a file called `openrc.sh` which will looks something like this:

```
#!/bin/bash

# With the addition of Keystone, to use an openstack cloud you should
# authenticate against keystone, which returns a **Token** and **Service
# Catalog**. The catalog contains the endpoint for all services the
# user/tenant has access to - including nova, glance, keystone, swift.
#
# *NOTE*: Using the 2.0 *auth api* does not mean that compute api is 2.0. We
# will use the 1.1 *compute api*
export OS_AUTH_URL=http://203.0.113.10:5000/v2.0

# With the addition of Keystone we have standardized on the term **tenant**
# as the entity that owns the resources.
export OS_TENANT_ID=98333aba48e756fa8f629c83a818ad57
export OS_TENANT_NAME="test-project"

# In addition to the owning entity (tenant), openstack stores the entity
# performing the action as the **user**.
export OS_USERNAME=test-user

# With Keystone you pass the keystone password.
echo "Please enter your OpenStack Password: "
read -s OS_PASSWORD_INPUT
export OS_PASSWORD=$OS_PASSWORD_INPUT
```

Note that this does not save your password in plain text, which is a good thing. But when you source or execute the script it will prompt for your password and then store your response in the environment variable `OS_PASSWORD`. It is important to note that this does require interactivity. It is possible to store a value directly in the script if you require a non interactive operation, but you then need to be extremely cautious with the security and permissions of this file.

EC2 compatibility credentials can be downloaded from the "EC2 Credentials" link in the left hand navigation bar, then selecting the project you want credentials for and clicking "Download EC2 Credentials". This will generate a zip file with server x509 certificates and a shell script fragment. Create a new directory in a secure location as, unlike the default `openrc`, these are live credentials containing all the authentication information required to access your cloud identity. Extract the zip file here. You should have cacert.pem, cert.pem, ec2rc.sh and pk.pem. The `ec2rc.sh` will be similar to this:

```
#!/bin/bash

NOVARC=$(readlink -f "${BASH_SOURCE:-${0}}" 2>/dev/null) ||\
 NOVARC=$(python -c 'import os,sys; print os.path.abspath(os.path.realpath(sys.argv[1]))'
"${BASH_SOURCE:-${0}}")
NOVA_KEY_DIR=${NOVARC%/*}
export EC2_ACCESS_KEY=df7f93ec47e84ef8a347bbb3d598449a
export EC2_SECRET_KEY=ead2fff9f8a344e489956deacd47e818
export EC2_URL=http://203.0.113.10:8773/services/Cloud
export EC2_USER_ID=42 # nova does not use user id, but bundling requires it
export EC2_PRIVATE_KEY=${NOVA_KEY_DIR}/pk.pem
export EC2_CERT=${NOVA_KEY_DIR}/cert.pem
export NOVA_CERT=${NOVA_KEY_DIR}/cacert.pem
export EUCALYPTUS_CERT=${NOVA_CERT} # euca-bundle-image seems to require this set

alias ec2-bundle-image="ec2-bundle-image --cert $EC2_CERT --privatekey $EC2_PRIVATE_KEY --user 42
--ec2cert $NOVA_CERT"
alias ec2-upload-bundle="ec2-upload-bundle -a $EC2_ACCESS_KEY -s $EC2_SECRET_KEY --url $S3_URL --
ec2cert $NOVA_CERT"
```

To put the the EC2 credentials into your environment source the `ec2rc.sh` file.

Command Line Tricks and Traps

The command line tools can be made to show the OpenStack API calls it's making by passing it the `--debug` flag for example:

```
# nova --debug list
```

This will show the HTTP requests from the client and the responses from the endpoints which can be helpful in creating custom tools written to the OpenStack API.

Keyring Support (https://wiki.openstack.org/wiki/KeyringSupport) can be a source of confusion to the point that, as of the time of this writing, there is a bug report (https://bugs.launchpad.net/python-novaclient/+bug/1020238) which has been open, closed as invalid, and reopened through a few cycles. The issue is that under some conditions the command line tools try to use a Python keyring as a credential cache and, under a subset of those conditions, another condition can arise where the tools prompt for a keyring password on each

use. If you find yourself in this unfortunate subset, adding the `--no-cache` flag or setting the environment variable `OS_NO_CACHE=1` will avoid the credentials cache. A warning note that this will cause the command line tool to authenticate on each and every interaction with the cloud.

cURL

Underlying the use of the command line tools is the OpenStack API. This is a RESTful API that runs over HTTP. There may be cases where you want to interact with the API directly or need to use it because of a suspected bug in one of the CLI tools. The best way to do this is use a combination of cURL (http://curl.haxx.se/) and another tool to parse the JSON, such as jq (http://stedolan.github.com/jq/), from the responses.

The first thing you will need to do is authenticate with the cloud using your credentials to get a token. Your credentials are a combination of username, password, and tenant (project). You can extract these values from the `openrc.sh` discussed above. The token will allow you to interact with your other service endpoints without needing to re-authenticate for every request. Tokens are typically good for 24 hours, and when the token expires, you will find out with a 401 (Unauthorized) response and can request another token.

1. Begin by having a look at your OpenStack service catalog.

```
$ curl -s -X POST http://203.0.113.10:35357/v2.0/tokens -d '{"auth": {"passwordCredentials":
{"username":"test-user", "password":"test-password"}, "tenantName":"test-project"}}' -H "Content-
type: application/json" | jq .
```

Read through the JSON response to get a feel for how the catalog is laid out.

2. To make working with subsequent requests easier, store the token in an environment variable.

```
$ TOKEN=`curl -s -X POST http://203.0.113.10:35357/v2.0/tokens -d '{"auth":
{"passwordCredentials": {"username":"test-user", "password":"test-password"}, "tenantName":"test-
project"}}' -H "Content-type: application/json" |  jq -r .access.token.id`
```

Now you can refer to your token on the command line as $TOKEN.

3. Pick a service endpoint from your service catalog, such as compute, and try out a request like listing instances (servers).

```
$ curl -s -H "X-Auth-Token: $TOKEN"
http://203.0.113.10:8774/v2/98333aba48e756fa8f629c83a818ad57/servers | jq .
```

To discover how API requests should be structured, read the OpenStack API Reference (http://api.openstack.org/api-ref.html). To chew through the responses using jq, refer to the jq Manual (http://stedolan.github.com/jq/manual/).

The `-s flag` used in the cURL commands above are used to prevent the progress meter from being shown. If you are having trouble running cURL commands, you'll want to remove it. Likewise, to help you troubleshoot cURL commands you can include the `-v` flag to show you the verbose output. There are many more extremely useful features in cURL, refer to the man page for all of the options.

Servers and Services

As an administrator, there are a few ways to discover what your OpenStack cloud looks like simply by using the OpenStack tools available. This section gives you an idea of how to get an overview of your cloud, its shape, size, and current state.

First, you can discover what servers belong to your OpenStack cloud by running:

```
# nova-manage service list | sort
```

The output will look similar to the following:

```
Binary           Host               Zone Status  State Updated_At
nova-cert        cloud.example.com nova enabled  :-)  2013-02-25 19:32:38
nova-compute     c01.example.com   nova enabled  :-)  2013-02-25 19:32:35
nova-compute     c02.example.com   nova enabled  :-)  2013-02-25 19:32:32
nova-compute     c03.example.com   nova enabled  :-)  2013-02-25 19:32:36
nova-compute     c04.example.com   nova enabled  :-)  2013-02-25 19:32:32
nova-compute     c05.example.com   nova enabled  :-)  2013-02-25 19:32:41
nova-consoleauth cloud.example.com nova enabled  :-)  2013-02-25 19:32:36
nova-network     cloud.example.com nova enabled  :-)  2013-02-25 19:32:32
nova-scheduler   cloud.example.com nova enabled  :-)  2013-02-25 19:32:33
```

The output shows that there are five compute nodes and one cloud controller. You see a smiley face like `:-)` which indicates that the services are up and running and functional. If a service is no longer available, the `:-)` will change to an `xxx`. This is an indication that you should troubleshoot why the service is down.

If you are still using `nova-volume` (this service is deprecated after the Folsom release), you might also see entries for the nova-volume service.

If you are using Cinder, you can see a similar list by running:

```
# cinder-manage host list | sort

host              zone
c01.example.com   nova
c02.example.com   nova
c03.example.com   nova
c04.example.com   nova
c05.example.com   nova
cloud.example.com nova
```

With these two tables, you now have a good overview of what servers and services make up your cloud.

You can also use the Identity Service (Keystone), to see what services are available in your cloud as well as what endpoints have been configured for the services.

The following commands will require you to have your shell environment configured with the proper administrative variables.

```
# keystone service-list

+-----+----------+----------+----------------------------+
| id  | name     | type     | description                |
+-----+----------+----------+----------------------------+
| ... | cinder   | volume   | Cinder Service             |
| ... | glance   | image    | OpenStack Image Service    |
| ... | nova_ec2 | ec2      | EC2 Service                |
| ... | keystone | identity | OpenStack Identity Service |
| ... | nova     | compute  | OpenStack Compute Service  |
+-----+----------+----------+----------------------------+
```

The output above shows that there are five services configured.

To see the endpoint of each service, run:

```
# keystone endpoint-list
```

```
+-----+-----------------------------------------+-/+-----------------------------------------+
| id  |            publicurl                    | \|               adminurl                 |
+-----+-----------------------------------------+-/+-----------------------------------------+
| ... | http://example.com:8774/v2/%(tenant_id)s| \| http://example.com:8774/v2/%(tenant_id)s|
| ... | http://example.com:8773/services/Cloud  | /| http://example.com:8773/services/Admin  |
| ... | http://example.com:9292/v1              | \| http://example.com:9292/v1              |
| ... | http://example.com:5000/v2.0            | /| http://example.com:35357/v2.0           |
| ... | http://example.com:8776/v1/%(tenant_id)s| \| http://example.com:8776/v1/%(tenant_id)s|
+-----+-----------------------------------------+-/+-----------------------------------------+
```

There should be a one-to-one mapping between a *service* and *endpoint*. Note the different URLs and ports between the public URL and the admin URL for some services.

Network

Next, take a look at what Fixed IP networks are configured in your cloud:

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

```
id IPv4        IPv6 start address DNS1 DNS2 VlanID project   uuid
1  10.1.0.0/24 None 10.1.0.3      None None 300    2725bbd   beacb3f2
2  10.1.1.0/24 None 10.1.1.3      None None 301    none      d0b1a796
```

This output shows that two networks are configured, each network containing 255 IPs (a /24 subnet). The first network has been assigned to a certain project while the second network is still open for assignment. This network can be assigned manually or it will automatically be assigned when a project launches their first instance.

To find out if any Floating IPs are available in your cloud, run

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

```
2725bbd458e2459a8c1bd36be859f43f 1.2.3.4                                    None nova vlan20
None                             1.2.3.5 48a415e7-6f07-4d33-ad00-814e60b010ff nova vlan20
```

Here, two Floating IPs are available. The first has been allocated to a project while the other is unallocated.

Users and Projects

To see a list of projects that have been added to the cloud, run:

```
# keystone tenant-list
```

```
+-----+----------+---------+
| id  | name     | enabled |
+-----+----------+---------+
| ... | jtopjian | True    |
| ... | alvaro   | True    |
| ... | everett  | True    |
| ... | admin    | True    |
| ... | services | True    |
| ... | jonathan | True    |
| ... | lorin    | True    |
| ... | anne     | True    |
| ... | rhulsker | True    |
| ... | tom      | True    |
| ... | adam     | True    |
+-----+----------+---------+
```

To see a list of users, run:

```
# keystone user-list
```

```
+-----+----------+---------+-------------------------------+
| id  | name     | enabled | email                         |
+-----+----------+---------+-------------------------------+
| ... | everett  | True    | everett.towne@backspace.com   |
| ... | jonathan | True    | jon@sfcu.edu                  |
| ... | nova     | True    | nova@localhost                |
| ... | rhulsker | True    | ryan.hulkster@cyberalbert.ca  |
| ... | lorin    | True    | lorinhoch@nsservices.com      |
| ... | alvaro   | True    | Alvaro.Perry@cyberalbert.ca   |
| ... | anne     | True    | anne.green@backspace.com      |
| ... | admin    | True    | root@localhost                |
| ... | cinder   | True    | cinder@localhost              |
| ... | glance   | True    | glance@localhost              |
```

```
| ... | jtopjian | True    | joe.topjian@cyberalbert.com  |
| ... | adam     | True    | adam@ossmanuals.net          |
| ... | tom      | True    | fafield@univm.edu.au         |
+-----+----------+---------+------------------------------+
```

Note two things. Sometimes a user and a group have a one-to-one mapping. This can happen if only one user will ever be part of a group. Also, there will always be some standard system accounts such as cinder, glance, nova, and swift.

Running Instances

To see a list of running instances, run:

```
# nova list --all-tenants
```

```
+-----+------------------+--------+-----------------------------------------+
| ID  | Name             | Status | Networks                                |
+-----+------------------+--------+-----------------------------------------+
| ... | Windows          | ACTIVE | novanetwork_1=10.1.1.3, 199.116.232.39    |
| ... | cloud controller | ACTIVE | novanetwork_0=10.1.0.6; jtopjian=10.1.2.3 |
| ... | compute node 1   | ACTIVE | novanetwork_0=10.1.0.4; jtopjian=10.1.2.4 |
| ... | devbox           | ACTIVE | novanetwork_0=10.1.0.3                    |
| ... | devstack         | ACTIVE | novanetwork_0=10.1.0.5                    |
| ... | initial          | ACTIVE | nova_network=10.1.7.4, 10.1.8.4           |
| ... | lorin-head       | ACTIVE | nova_network=10.1.7.3, 10.1.8.3           |
+-----+------------------+--------+-----------------------------------------+
```

Unfortunately this command will not tell you various details about the running instances, such as what compute node the instance is running on, what flavor the instance is, etc. You can use the following command to view details about individual instances:

```
# nova show <uuid>
```

For example:

```
# nova show 81db556b-8aa5-427d-a95c-2a9a6972f630

+-------------------------------------+----------------------------------+
| Property                            | Value                            |
+-------------------------------------+----------------------------------+
| OS-DCF:diskConfig                   | MANUAL                           |
| OS-EXT-SRV-ATTR:host                | c02.example.com                  |
| OS-EXT-SRV-ATTR:hypervisor_hostname | c02.example.com                  |
| OS-EXT-SRV-ATTR:instance_name       | instance-00000029                |
| OS-EXT-STS:power_state              | 1                                |
| OS-EXT-STS:task_state               | None                             |
| OS-EXT-STS:vm_state                 | active                           |
| accessIPv4                          |                                  |
| accessIPv6                          |                                  |
| config_drive                        |                                  |
| created                             | 2013-02-13T20:08:36Z             |
| flavor                              | m1.small (6)                     |
| hostId                              | ...                              |
| id                                  | ...                              |
| image                               | Ubuntu 12.04 cloudimg amd64 (...) |
| key_name                            | jtopjian-sandbox                 |
| metadata                            | {}                               |
| name                                | devstack                         |
| novanetwork_0 network               | 10.1.0.5                         |
| progress                            | 0                                |
| security_groups                     | [{u'name': u'default'}]          |
| status                              | ACTIVE                           |
| tenant_id                           | ...                              |
| updated                             | 2013-02-13T20:08:59Z             |
| user_id                             | ...                              |
+-------------------------------------+----------------------------------+
```

Managing Projects and Users

An OpenStack cloud does not have much value without users. This chapter covers topics that relate to managing users, projects, and quotas.

Projects or tenants?

In OpenStack user interfaces and documentation, you will sometimes see the term "project" to refer to a group of users, and other times you will see the term "tenant". These terms are interchangeable.

The reason behind this bifurcation is historical. The initial implementation of the OpenStack Compute Service (nova) had its own authentication system and used the term "project". When authentication was broken out into the OpenStack Identity Service (keystone) project, the new project used the term "tenant" to refer to a group of users. Because of this legacy, some of the OpenStack tools refer to "projects" and some refer to "tenants".

In this guide, we will use the term "project" unless we are showing an example interaction with a tool that uses the term "tenant".

Managing Projects

Users must be associated with at least one project, though they may belong to many. Therefore, you should add at least one project before adding users.

Adding Projects

To create a project via Dashboard:

1. Log in as an administrative user.
2. Select the "Projects" link in the left hand navigation bar.
3. Click on the "Create Project" button at the top right.

This will pop up a form asking from asking for a project name and an optional, but recommended, description. There is also a check box at the bottom of the form to enable this project, by default this is enabled.