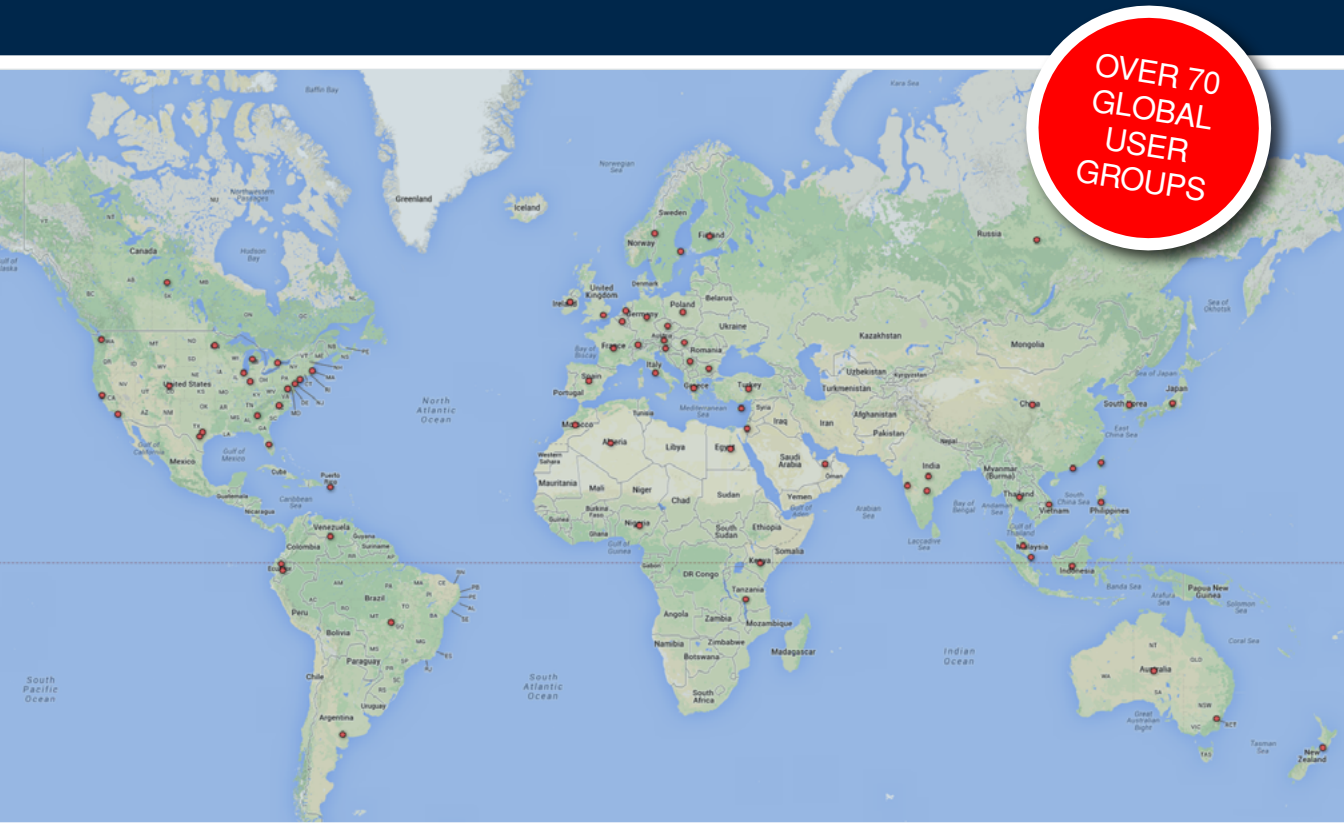
O'REILLY®

OpenStack Operations Guide

SET UP AND MANAGE YOUR OPENSTACK CLOUD

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

Join the global community!



Get Involved and get more out of OpenStack!

- Take the **User Survey** and influence the OpenStack Roadmap
- Find a local **User Group** near you and attend a meet up
- Attend a **Training Course**

openstack
CLOUD SOFTWARE

OpenStack Operations Guide

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

Beijing • Cambridge • Farnham • Köln • Sebastopol • Tokyo O'REILLY®

OpenStack Operations Guide

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

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (*http://my.safaribooksonline.com*). For more information, contact our corporate/institutional sales department: 800-998-9938 or corporate@oreilly.com.

Editors: Andy Oram and Brian Anderson

Interior Designer: David Futato

Cover Designer: Karen Montgomery

March 2014: First Edition

See *http://oreilly.com/catalog/errata.csp?isbn=9781491946954* for release details.

978-1-491-94695-4

[LSI]

Table of Contents

Acknowledgments

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 the Rackspace Austin office. The authors are all members of the OpenStack Foundation, which you can join. Go to the Foundation web site at `http://openstack.org/join`.

We 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.
- 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 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.

Inside the book sprint room with us each day was our book sprint facilitator Adam Hyde. Without his tireless support and encouragement, we would have thought a book of this scope was impossible in five days. Adam has proven the book sprint

method effectively again and again. He creates both tools and faith in collaborative authoring at http://www.booksprints.net/.

We couldn't have pulled it off without so much supportive help and encouragement.

Document Change History

This version of the document replaces and obsoletes all previous versions. The following table describes the most recent changes:

Revision Date	Summary of Changes
2013-05-13	• Updated description of availability zones.
2013-04-02	• Fixes to ensure samples fit in page size and notes are formatted.
2013-03-22	• Stopped chunking in HTML output.
2013-03-20	• Editorial changes. • Added `glossterm` tags to glossary terms. • Cleaned up formatting in code examples. • Removed future tense.
2013-03-11	• Moved files to OpenStack github repository.

Preface

OpenStack is an open source platform that lets you build an Infrastructure as a Service (IaaS) cloud that runs on commodity hardware.

Introduction to OpenStack

OpenStack believes in open source, open design, open development, all in an open community so anyone can participate. The long-term vision for OpenStack is to produce a ubiquitous open source cloud computing platform that meets the needs of public and private cloud providers regardless of size. OpenStack services control large pools of compute, storage, and networking resources throughout a data center.

Each service provides a REST API so that all these resources can be managed through a dashboard that gives administrators control while empowering users to provision resources through a web interface, through a command-line client, or through software development kits that support the API. Many OpenStack APIs are extensible, meaning you can keep compatibility with a core set of calls while providing access to more resources and innovating through API extensions. The OpenStack project is a global collaboration of developers and cloud computing technologists producing the open standard cloud computing platform for both public and private clouds. The project aims to deliver solutions for all types of clouds by being simple to implement, massively scalable, and feature-rich. The technology consists of a series of interrelated projects delivering various components for a cloud infrastructure solution.

Okay, you say, OpenStack is open source cloud. Are there others? Yes, Eucalyptus offers an open source version of their private cloud offering that provides services compatible with Amazon Web Services (AWS) APIs such as EC2 and S3. With Ubuntu Server 11.10 release, OpenStack became supported by default rather than Eucalyptus for Ubuntu Server releases. The product targets private cloud deployers who want to maintain compatibility with Amazon Web Services by partnering with Amazon. Apache CloudStack is another example of an open source cloud project, but it only runs on Ubuntu 10.04. It offers computing as a core service, with connections to stor-

age services. Citrix purchased the project technology for $200 million in 2011 when buying cloud.com, then released it to Apache in April 2012. The API compatibility for CloudStack centers on AWS APIs.

OpenStack is 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. OpenStack is more than a software package that you run as-is. It lets you integrate a number of different technologies to construct a cloud. This approach provides great flexibility, but the number of options might be bewildering at first.

Who This Book Is For

This guide assumes that you are familiar with the Ubuntu distribution of Linux, SQL databases, and virtualization. You must be comfortable administering and configuring multiple Linux machines for networking. You must install and maintain a MySQL database, and occasionally run SQL queries against it.

One of the most complex aspects of an OpenStack cloud is the networking configuration. You should be familiar with concepts such as DHCP, Linux bridges, VLANs, and iptables. You must also have access to a network hardware expert who can configure the switches and routers required in your OpenStack cloud.

This book is for those of you starting to run OpenStack clouds as well as those of you who were handed a running one and want to keep it running well. Perhaps you're on a devops team, perhaps you are a system administrator starting to dabble in the cloud, or maybe you want to get on that OpenStack cloud team at your company. This book is for all of you.

How This Book Is Organized

This book is organized in two parts, the architecture decisions for designing OpenStack clouds and the repeated operations for running OpenStack clouds.

Chapter 1: Provisioning and Deployment: While this book doesn't describe installation, we do recommend automation for deployment and configuration, discussed in this chapter.

Chapter 2: Cloud Controller Design: The cloud controller is an invention for the sake of consolidating and describing which services run on which nodes. The chapter discusses hardware and network considerations as well as how to design the cloud controller for performance and separation of services.

Chapter 3: Scaling: This chapter discusses the growth of your cloud resources through scaling and segregation considerations.

Chapter 4: Compute Nodes: This chapter describes the compute nodes, which are dedicated to run virtual machines. Some hardware choices come into play here as well as logging and networking descriptions.

Chapter 5: Storage Decisions: Along with other architecture decisions, storage concepts within OpenStack take a lot of consideration, and this chapter lays out the choices for you.

Chapter 6: Network Design: Your OpenStack cloud networking needs to fit into your existing networks while also enabling the best design for your users and administrators, and this chapter gives you in-depth information about networking decisions.

Chapter 7: Example Architecture: Because of all the decisions the previous chapters discuss, this chapter describes the decisions made for this particular book and much of the justification for the example architecture.

Chapter 8: Lay of the Land: This chapter is written to let you get your hands wrapped around your OpenStack cloud through command line tools and understanding what is already set up in your cloud.

Chapter 9: Managing Projects and Users: This chapter walks through those user-enabling processes that all admins must face to manage users, give them quotas to parcel out resources, and so on.

Chapter 10: User-facing Operations: This chapter moves along to show you how to use OpenStack cloud resources and train your users as well.

Chapter 11: Maintenance, Failures, and Debugging: This chapter goes into the common failures the authors have seen while running clouds in production, including troubleshooting.

Chapter 12: Network Troubleshooting: Because network troubleshooting is especially difficult with virtual resources, this chapter is chock-full of helpful tips and tricks to tracing network traffic, finding the root cause of networking failures, and debugging related services like DHCP and DNS.

Chapter 13: Logging and Monitoring: This chapter shows you where OpenStack places logs and how to best to read and manage logs for monitoring purposes.

Chapter 14: Backup and Recovery: This chapter describes what you need to back up within OpenStack as well as best practices for recovering backups.

Chapter 15: Customize: When you need to get a specialized feature into OpenStack, this chapter describes how to use DevStack to write custom middleware or a custom scheduler to rebalance your resources.

Chapter 16: Upstream OpenStack: Because OpenStack is so, well, open, this chapter is dedicated to helping you navigate the community and find out where you can help and where you can get help.

Chapter 17: Advanced Configuration: Much of OpenStack is driver-oriented, where you can plug in different solutions to the base set of services. This chapter describes some advanced configuration topics.

Appendix A: Use Cases: You can read a small selection of use cases from the OpenStack community with some technical detail and further resources.

Appendix B: Tales From the Cryp^H^H^H^H Cloud: These are shared legendary tales of image disappearances, VM massacres, and crazy troubleshooting techniques to share those hard-learned lessons and wisdom.

Appendix C: Resources: So many OpenStack resources are available online due to the fast-moving nature of the project, but there are also listed resources the authors found helpful while learning themselves.

Glossary: A list of terms used in this book is included, which is a subset of the larger OpenStack Glossary available online.

Why and How We Wrote This Book

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 to others. After months of being the point people for an OpenStack cloud, we also wanted to have a document to hand to our system administrators so they'd know how to operate the cloud on a daily basis — both reactively and proactively. We wanted to provide more detailed technical information about the decisions that deployers make along the way.

We wrote this book to help you:

- Design and create an architecture for your first non-trivial OpenStack cloud. After you read this guide, you'll know which questions to ask and how to organize your compute, networking, storage resources, and the associated software packages.
- Perform the day-to-day tasks required to administer a cloud.

We wrote this book in a Book Sprint, which is a facilitated rapid development production method for books. For more information see the Book Sprint site. Your authors cobbled this book together in five days during February 2013, fueled by caffeine and the best take-out food that Austin, Texas could offer.

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 each other. 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:

- **Tom Fifield**. After learning about scalability in computing from particle physics experiments like ATLAS at the LHC, Tom worked on OpenStack clouds in production to support the Australian public research sector. Tom currently serves as an OpenStack community manager and works on OpenStack documentation in his spare time.
- **Diane Fleming**. Diane works on the OpenStack API documentation tirelessly. She helped out wherever she could on this project.
- **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**. An academic turned software developer-slash-operator, Lorin 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 technical architect 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 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 as a systems architect, and his experiences have generated a wealth of troubleshooting skills for cloud environments.

How to Contribute to 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 contribute to it. OpenStack documentation follows the coding principles of iterative work, with bug logging, investigating, and fixing. We also store the source content on Github and invite collaborators through the OpenStack Gerrit installation, which offers reviews. For the O'Reilly edition of this book, we are using their Atlas system which also stores source content on Github and enables collaboration among contributors.

Learn more about how to contribute to the OpenStack docs at Documentation How To (http://wiki.openstack.org/Documentation/HowTo).

If you find a bug and can't fix it or aren't sure it's really a doc bug, log a bug at OpenStack Manuals (http://bugs.launchpad.net/openstack-manuals). Tag the bug under Extra options with `ops-guide` tag to indicate that the bug is in this guide. You can assign the bug to yourself if you know how to fix it. Also, a member of the OpenStack doc-core team can triage the doc bug.

CHAPTER 1

Provisioning and Deployment

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 infrastructure includes systems to automatically install the operating system's initial configuration and 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, including 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:

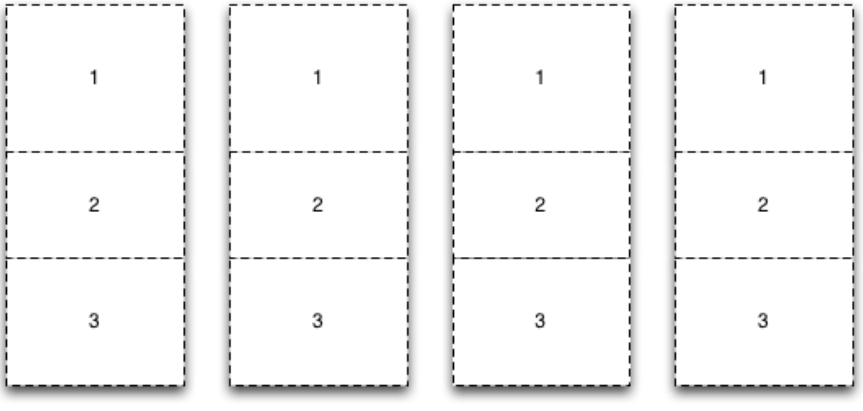
- File system
- Swap space

RAID is not used in this setup.

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 parti-

tion 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 is 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 on which only your cloud communicates.

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 data center 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.

CHAPTER 2

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*.

For more details about the overall architecture, see the Chapter 7.

As described in this guide, the cloud controller is a single node that hosts the databases, message queue service, authentication and authorization service, 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:

- The number of instances that you expect to run
- The number of compute nodes that you have
- The number of users who will access the compute or storage services
- Whether users interact with your cloud through the REST API or the dashboard
- Whether users authenticate against an external system (such as, LDAP or Active Directory)
- How long you expect single instances to run

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 report 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 that your messaging queue handles requests successfully and size accordingly.
How many users will access the API?	If many users will make multiple requests, make sure that 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 back-end 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 understood 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 virtualize or not. Some services, such as nova-compute, swift-proxy and swift-object servers, should not be virtualized. However, control servers can often be happily virtualized - 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. Loss of this ability leads 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.

Application Programming Interface (API)

All public access, whether direct, through a command line client, or through the web-based dashboard, uses the API service. Find the API reference at http://api.openstack.org/.

You must choose whether you want to support the Amazon EC2 compatibility APIs, or just the OpenStack APIs. One issue you might 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 set up 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.

You can use various techniques to handle 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 back-end. 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 back-end. Currently, it supports:

- OpenStack Object Storage. Allows you to store images as objects.
- File system. Uses any traditional file system to store the images as files.
- S3. Allows you to fetch images from Amazon S3.
- HTTP. Allows you to fetch images from a web server. You cannot write images by using this mode.

If you have an OpenStack Object Storage service, we recommend using this as a scalable place to store your images. You can also use a file system with sufficient performance or Amazon S3 - unless you do not need the ability to upload new images through 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 the policy.json file. For information on how to configure these, see Chapter 9.

The Identity Service supports different plugins for back-end 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

Because the cloud controller handles so many different services, it must be able to handle the amount of traffic that hits 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 is the network gateway for all instances, then the Cloud Controller must support the total amount of traffic that travels between your cloud and the public Internet.

We recommend that you use a fast NIC, such as 10 GB. You can also choose to use two 10 GB NICs and bond them together. While you might not be able to get a full bonded 20 GB speed, different transmission streams use different NICs. For example, if the Cloud Controller transfers two images, each image uses a different NIC and gets a full 10 GB of bandwidth.

CHAPTER 3

Scaling

If your cloud is successful, eventually you must 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 is required (`flavor disk size × number of instances`). You can use these ratios to determine how much additional infrastructure you need to support your cloud.

The default OpenStack flavors are:

Name	Virtual cores	Memory	Disk	Ephemeral
m1.tiny	1	512 MB	1 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

Assume that the following set-up supports (200 / 2) × 16 = 1600 VM instances and requires 80 TB of storage for `/var/lib/nova/instances`:

- 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

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 puts 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 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 Chapter 13.

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 you need	A single API endpoint for compute, or you require a second level of scheduling.	Discrete regions with separate API endpoints and no coordination between regions.	Logical separation within your nova deployment for physical isolation or redundancy.	To schedule a group of hosts with common features.
Example	A cloud with multiple sites where you can schedule VMs "anywhere" or on a particular site.	A cloud with multiple sites, where you schedule VMs to a particular site and you want a shared infrastructure.	A single site cloud with equipment fed by separate power supplies.	Scheduling to hosts with trusted hardware support.
Overhead	• A new service, `nova-cells` • 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 and regions), and those which merely divide a single deployment (availability zones and host aggregates).

Cells and Regions

OpenStack Compute cells are 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

You can use availability zones, host aggregates, or both to partition a nova deployment.

Availability zones are implemented through and configured in a similar way to host aggregates.

However, you use an availability zone and a host aggregate for different reasons:

- **Availability zone**. Enables you to arrange OpenStack Compute hosts into logical groups, and provides a form of physical isolation and redundancy from other availability zones, such as by using separate power supply or network equipment.

  You define the availability zone in which a specified Compute host resides locally on each server. An availability zone is commonly used to identify a set of servers that have a common attribute. For instance, if some of the racks in your data center are on a separate power source, you can put servers in those racks in their own availability zone. Availability zones can also help separate different classes of hardware.

  When users provision resources, they can specify from which availability zone they would like their instance to be built. This allows cloud consumers to ensure that their application resources are spread across disparate machines to achieve high availability in the event of hardware failure.
- **Host aggregate**. Enables you to partition OpenStack Compute deployments into logical groups for load balancing and instance distribution. You can use host aggregates to further partition an availability zone. For example, you might use host aggregates to partition an availability zone into groups of hosts that either share common resources, such as 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, you might use a host aggregate to group a set of hosts that share specific flavors or images.

Previously, all services had an availability zone. Currently, only the nova-compute service has its own availability zone. Services such as nova-scheduler, nova-network, nova-conductor have always spanned all availability zones.

When you run any of the following operations, the services appear in their own internal availability zone (CONF.internal_service_availability_zone):

- nova host-list (os-hosts)
- euca-describe-availability-zones verbose
- nova-manage service list

The internal availability zone is hidden in euca-describe-availability_zones (non-verbose).

CONF.node_availability_zone has been renamed to CONF.default_availability_zone and is only used by the nova-api and nova-scheduler services.

CONF.node_availability_zone still works but is deprecated.

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 remains 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 to support instance migration.

The typical hardware recommended for use with OpenStack is the 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 spend more, if you have existing servers, provided they meet your performance requirements and virtualization 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.

CHAPTER 4

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 depends 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 configuration reference (http://docs.openstack.org/trunk/config-reference/content/section_compute-hypervisors.html).

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 must 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 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 option is that it scales to external storage when you require additional storage.

However, this option has 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 are 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.

Live migration can be also done with non-shared storage, using a feature known as *KVM live block migration*. While an earlier implementation of block-based migration in KVM and QEMU was considered unreliable, there is a newer, more reliable implementation of block-based live migration as of QEMU 1.4 and libvirt 1.0.2 that is also compatible with OpenStack. However, none of the authors of this guide have first-hand experience using live block migration.

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 allocates up to 16 virtual cores on a node per physical core. For example, if a physical node has 12 cores, the scheduler allocates up to 192 virtual cores to instances (such as, 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 allocates 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 allocates instances to that node until the sum of the RAM associated with the instances reaches 72 GB (such as nine instances, in the case where each instance has 8 GB of RAM).

You must select the appropriate CPU and RAM allocation ratio for your particular use case.

Logging

Logging is detailed more fully in the section called "Logging". 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 (such as logstash).

Networking

Networking in OpenStack is a complex, multi-faceted challenge. See Chapter 6.

CHAPTER 5

Storage Decisions

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 can configure with your cloud.

OpenStack Storage Concepts

	Ephemeral storage	Block storage	Object storage
Used to…	Run operating system and scratch space	Add additional persistent storage to a virtual machine (VM)	Store data, including VM images
Accessed through…	A file system	A block device that can be partitioned, formatted and mounted (such as, /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

	Ephemeral storage	Block storage	Object storage
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…	10 GB first disk, 30GB second disk	1 TB disk	10s of TBs of dataset storage

If you only deploy the OpenStack Compute Service (nova), your users do not have access to any form of persistent storage by default. The disks associated with VMs are "ephemeral", meaning that (from the user's point of view) they effectively disappear when a virtual machine is terminated. You must 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 want to provide them with object storage. In addition, OpenStack can store your virtual machine (VM) 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 VM instances.

These volumes are persistent: they can be detached from one instance and reattached to another, and the data remains intact. Block storage is implemented in OpenStack by the OpenStack Block Storage (Cinder) project, which supports multiple back-ends in the form of drivers. Your choice of a storage back-end must be supported by a Block Storage driver.

Most block storage drivers allow the instance to have direct access to the underlying storage hardware's block device. This helps increase the overall read/write IO.

Experimental support for utilizing files as volumes began in the Folsom release. This initially started as a reference driver for using NFS with Cinder. By Grizzly's release, this has expanded into a full NFS driver as well as a GlusterFS driver.

These drivers work a little differently than a traditional "block" storage driver. On an NFS or GlusterFS file system, a single file is created and then mapped as a "virtual" volume into the instance. This mapping/translation is similar to how OpenStack utilizes QEMU's file-based virtual machines stored in `/var/lib/nova/instances`.

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 Back-ends

In general, when you select storage back-ends, ask the following questions:

- 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?

To deploy your storage by using entirely commodity hardware, you can use a number of open-source packages, as shown in the following table:

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

* This list of open-source file-level shared storage solutions is not exhaustive, other open source solutions exist (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 Back-end Technologies

This section provides a high-level overview of the differences among the different commodity storage back-end technologies.

- **OpenStack Object Storage (Swift)**. 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**. 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 Cinder 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. Ceph also supports keystone-based authentication (as of version 0.56), so it can be a seamless swap in for the default OpenStack Swift implementation.

  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**. 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. The 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) back-end 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 are created from LVM logical volumes.

LVM does *not* provide any replication. Typically, administrators configure RAID on nodes that use LVM as block storage 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 (such as, ext3, ext4, xfs, btrfs). ZFS has a number of advantages over ext4, including improved data integrity checking.

The ZFS back-end 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 back-end for Block Storage requires a Solaris-based operating system and we assume that your experience is primarily with Linux-based systems.

- **Sheepdog**. 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 fails, 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 exists - 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 might seem unfamiliar at first. Consider these main traffic flows:

- Among object, container, and account servers
- Between those servers and the proxies
- Between the proxies and your users

Object Storage is very 'chatty' among servers hosting data - even a small cluster does 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, replication of the object begins.

Consider the scenario where an entire server fails, and 24 TB of data needs to be transferred "immediately" to remain at three 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, which means that 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.

CHAPTER 6

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, you must plan the number of IP addresses that you need for both your guest instances as well as management infrastructure. Additionally, you must research and discuss cloud network connectivity through proxy servers and firewalls.

Management Network

A management network, typically consisting of a separate switch and separate NICs, is a recommended option. This segregation prevents system administration and monitoring system access from being disrupted by traffic generated by the guests themselves.

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 address-

es 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 is external to the compute node, this network is 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 (such as, 1 GB) interface on compute or storage nodes is used for system administrators or monitoring tools to access the host instead of going through 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. To allow OpenStack to efficiently use VLANs, you must 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 must support a max of 100 projects, pick a free VLAN range that your network infrastructure is currently not using (such as, VLAN 200 - 299). You must 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 uses up an entire subnet or VLAN. This decrements 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 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 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 cannot communicate on the network. This is not true with multi-host, but multi-host requires that each compute node has a public IP address 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.

CHAPTER 7

Example Architecture

Because OpenStack is highly configurable, with many different back-ends and network configuration options, it is difficult to write documentation that covers all possible OpenStack deployments. Therefore, this guide defines 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) back-end	file
Identity Service (keystone) driver	SQL
Block Storage Service (cinder) back-end	LVM/iSCSI
Live Migration back-end	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.

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

- dashboard
- block storage
- floating IP addresses
- 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 back-end or configuration, we did not consider it for the example architecture. We believe that many clouds that currently run OpenStack in production have made similar choices.

You must first choose the operating system that runs on all of the physical nodes. While OpenStack is supported on several distributions of Linux, we used **Ubuntu 12.04 LTS (Long Term Support)**, which is used by the majority of the development community, has feature completeness compared with other distributions, and has 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 allows you to upgrade to future OpenStack releases while 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 back-end in the OpenStack Image Catalog and Delivery Service (Glance). If the cloud you are designing also intends to run Object Storage, it is trivial to enable this as the back-end instead, and a recommended approach.

We chose the **SQL back-end for Identity Service (keystone)** over others, such as LDAP. This back-end is simple to install and is robust. The authors acknowledge that many installations want to bind with existing directory services, and caution careful understanding of the array of options available (http://docs.openstack.org/trunk/config-reference/content/ch_configuring-openstack-identity.html#configuring-keystone-for-ldap-backend)

The Block Storage service (cinder) is installed natively on external storage nodes and uses 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 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 Use 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.

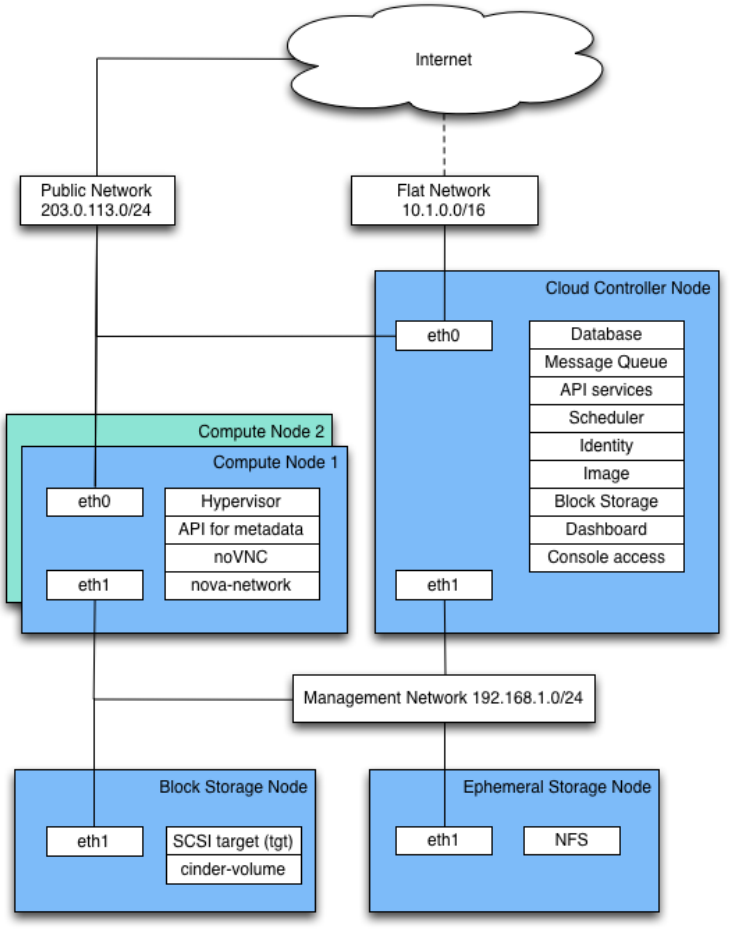
Why Use Multi-host Networking?

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, you cannot access your instances and the instances cannot access the Internet. The single node that runs the nova-network service can become a bottleneck if excessive network traffic comes in and goes 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

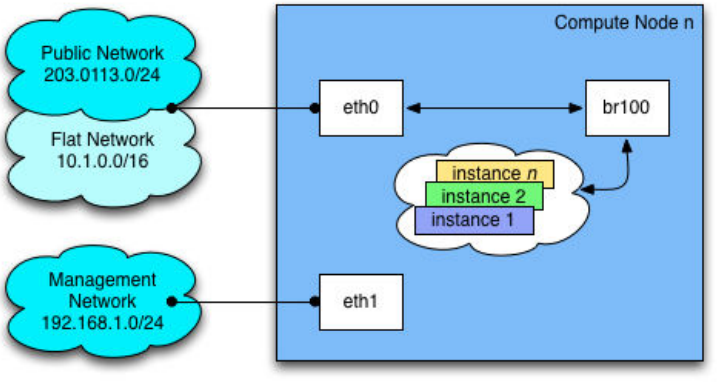
The reference architecture consists of multiple compute nodes, a cloud controller, an external NFS storage server for instance storage and an OpenStack Block Storage server for volume storage. A network time service (Network Time Protocol, NTP) synchronizes time for all the nodes. FlatDHCPManager in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



Optional Extensions

You can extend this reference architecture as follows:

- Add additional cloud controllers (see Chapter 11).
- Add an OpenStack Storage service (http://docs.openstack.org/havana/install-guide/install/apt/content/ch_swift.html)
- Add additional OpenStack Block Storage hosts (see Chapter 11).

What's Next?

Congratulations! By now, you should have a solid design for your cloud. We now recommend that you turn to the OpenStack Install and Deploy Manual - Ubuntu (http://docs.openstack.org/havana/install-guide/install/apt/), 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 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 is useful to read in advance, so that you can put best practices into effect to simplify your life in the long run. Other content is more useful as a reference that you might refer when an unexpected event occurs, such a power failure or troubleshooting a particular problem.

CHAPTER 8

Lay of the Land

From this point forward in the guide, we assume that you have an OpenStack cloud up and running. This section helps you set 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. We highlight those differences.

We strongly suggest that you install 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, install 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 must specify a name for the Python egg that is 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 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 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 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 to access the user settings page where you can set your language and timezone preferences for the dashboard view. More importantly, this action changes the left hand navigation column to include OpenStack API and EC2 Credentials links, which let 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 lists 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 Download RC. This generates a file called `openrc.sh`, which 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 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
```

This does not save your password in plain text, which is a good thing. But when you source or run the script, it prompts for your password and then stores 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 generates 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` is 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

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 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 example shows 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` avoids the credentials cache.

This causes 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, which 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 must do is authenticate with the cloud using your credentials to get an authentication 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 allows 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 are alerted with a 401 (Unauthorized) response and you can request another token.

1. 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 .
```

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

   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, see 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 looks like 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 `:-)` changes to an `XXX`. This is an indication that you should troubleshoot why the service is down.

If you are using Cinder, run the following command to see a similar listing:

```
$ 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 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
```

```
---+-----------------------------------------+--
   |                publicurl                |
---+-----------------------------------------+--
   | http://example.com:8774/v2/%(tenant_id)s |
   |          http://example.com:9292         |
   |        http://example.com:8000/v1        |
   |       http://example.com:5000/v2.0       |
---+-----------------------------------------+--

---+-----------------------------------------+--
   |                 adminurl                |
---+-----------------------------------------+--
   | http://example.com:8774/v2/%(tenant_id)s |
   |          http://example.com:9292         |
   |        http://example.com:8000/v1        |
   |       http://example.com:5000/v2.0       |
---+-----------------------------------------+--
```

This example shows two columns pulled from the larger listing. 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.

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

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

Diagnose your compute nodes

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

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

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

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

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

```
+------------------+------------+
| Property         | Value      |
+------------------+------------+
| cpu0_time        | 2870000000 |
| memory           | 524288     |
| vda_errors       | -1         |
| vda_read         | 262144     |
| vda_read_req     | 112        |
| vda_write        | 5606400    |
| vda_write_req    | 376        |
| vnet0_rx         | 63343      |
```

```
| vnet0_rx_drop    | 0          |
| vnet0_rx_errors  | 0          |
| vnet0_rx_packets | 431        |
| vnet0_tx         | 4905       |
| vnet0_tx_drop    | 0          |
| vnet0_tx_errors  | 0          |
| vnet0_tx_packets | 45         |
+------------------+------------+
```

Network

Next, take a look at what Fixed IP networks are configured in your cloud. You can use the *nova* command-line client to get the IP ranges.

```
$ nova network-list
+--------------------------------------+--------+--------------+
| ID                                   | Label  | Cidr         |
+--------------------------------------+--------+--------------+
| 3df67919-9600-4ea8-952e-2a7be6f70774 | test01 |  10.1.0.0/24 |
| 8283efb2-e53d-46e1-a6bd-bb2bdef9cb9a | test02 |  10.1.1.0/24 |
+--------------------------------------+--------+--------------+
```

The *nova-manage* tool can provide some additional details.

```
$ 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. You can assign this network manually or it is automatically 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        |
+-----+----------+---------+-----------------------------+
```

Sometimes a user and a group have a one-to-one mapping. This happens for standard system accounts, such as cinder, glance, nova, and swift, or when only one user is ever part of a group.

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 does not tell you various details about the running instances, such as what compute node the instance is running on, what flavor the instance is, and so on. 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                             | ...                              |
+-------------------------------------+----------------------------------+
```

CHAPTER 9

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, a group of users is referred to as a project or tenant. These terms are interchangeable.

The initial implementation of the OpenStack Compute Service (nova) had its own authentication system and used the term `project`. When authentication moved into the OpenStack Identity Service (keystone) project, it 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.

This guide uses the term `project`, unless an example shows 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 through the 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.

You are prompted for a project name and an optional, but recommended, description. Select the check box at the bottom of the form to enable this project. By default, this is enabled.

It is also possible to add project members and adjust the project quotas. We'll discuss those later, but in practice it can be quite convenient to deal with all these operations at one time.

To create a project through the command-line interface (CLI):

To add a project through the command line, you must use the keystone utility, which uses "tenant" in place of "project":

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

This command creates a project named "demo". Optionally, you can add a description string by appending `--description` *`tenant-description`* which can be very useful. You can also create a group in a disabled state by appending `--enabled false` to the command. By default, projects are created in an enabled state.

Quotas

To prevent system capacities from being exhausted without notification, you can set up quotas. Quotas are operational limits. For example, the number of gigabytes allowed per tenant can be controlled to ensure that a single tenant cannot consume all of the disk space. Quotas are currently enforced at the tenant (or project) level, rather than by user.

Using the command-line interface, you can manage quotas for the OpenStack Compute Service and the Block Storage Service.

Typically, default values are changed because a tenant requires more than the OpenStack default of 10 volumes per tenant, or more than the OpenStack default of 1TB of disk space on a Compute node.

To view all tenants, run:

```
$ keystone tenant-list
+----------------------------------+----------+---------+
|                id                |   name   | enabled |
+----------------------------------+----------+---------+
| a981642d22c94e159a4a6540f70f9f8d |  admin   |   True  |
| 934b662357674c7b9f5e4ec6ded4d0e7 | tenant01 |   True  |
| 7bc1dbfd7d284ec4a856ea1eb82dca80 | tenant02 |   True  |
| 9c554aaef7804ba49e1b21cbd97d218a | services |   True  |
+----------------------------------+----------+---------+
```

Set Compute Service Quotas

As an administrative user, you can update the Compute Service quotas for an existing tenant, as well as update the quota defaults for a new tenant.

Quota	Description	Property Name
Fixed Ips	Number of fixed IP addresses allowed per tenant. This number must be equal to or greater than the number of allowed instances.	`fixed-ips`
Floating Ips	Number of floating IP addresses allowed per tenant.	`floating-ips`
Injected File Content Bytes	Number of content bytes allowed per injected file.	`injected-file-content-bytes`
Injected File Path Bytes	Number of bytes allowed per injected file path.	`injected-file-path-bytes`
Injected Files	Number of injected files allowed per tenant.	`injected-files`
Instances	Number of instances allowed per tenant.	`instances`
Key Pairs	Number of key pairs allowed per user.	`key-pairs`
Metadata Items	Number of metadata items allowed per instance.	`metadata-items`
Ram	Megabytes of instance ram allowed per tenant.	`ram`
Security Group Rules	Number of rules per security group.	`security-group-rules`
Security Groups	Number of security groups per tenant.	`security-groups`
VCPUs	Number of instance cores allowed per tenant.	`cores`

View and update Compute quotas for a tenant (project)

As an administrative user, you can use the *nova quota-** commands, which are provided by the `python-novaclient` package, to view and update tenant quotas.

To view and update default quota values

1. List all default quotas for all tenants, as follows:

   ```
   $ nova quota-defaults
   ```

   For example:

   ```
   $ nova quota-defaults
   +-----------------------------+-------+
   | Property                    | Value |
   +-----------------------------+-------+
   | metadata_items              | 128   |
   | injected_file_content_bytes | 10240 |
   | ram                         | 51200 |
   | floating_ips                | 10    |
   | key_pairs                   | 100   |
   | instances                   | 10    |
   | security_group_rules        | 20    |
   | injected_files              | 5     |
   | cores                       | 20    |
   | fixed_ips                   | -1    |
   | injected_file_path_bytes    | 255   |
   | security_groups             | 10    |
   +-----------------------------+-------+
   ```

2. Update a default value for a new tenant, as follows:

   ```
   $ nova quota-class-update default key value
   ```

   For example:

   ```
   $ nova quota-class-update default instances 15
   ```

To view quota values for a tenant (project)

1. Place the tenant ID in a useable variable, as follows:

   ```
   $ tenant=$(keystone tenant-list | awk '/tenantName/ {print $2}')
   ```

2. List the currently set quota values for a tenant, as follows:

   ```
   $ nova quota-show --tenant $tenant
   ```

   For example:

   ```
   $ nova quota-show --tenant $tenant
   +-----------------------------+-------+
   | Property                    | Value |
   +-----------------------------+-------+
   | metadata_items              | 128   |
   | injected_file_content_bytes | 10240 |
   | ram                         | 51200 |
   | floating_ips                | 12    |
   | key_pairs                   | 100   |
   | instances                   | 10    |
   ```

```
| security_group_rules        | 20    |
| injected_files              | 5     |
| cores                       | 20    |
| fixed_ips                   | -1    |
| injected_file_path_bytes    | 255   |
| security_groups             | 10    |
+-----------------------------+-------+
```

To update quota values for a tenant (project)

1. Obtain the tenant ID, as follows:

```
$ tenant=$(keystone tenant-list | awk '/tenantName/ {print $2}')
```

2. Update a particular quota value, as follows:

```
# nova quota-update --quotaName quotaValue tenantID
```

   For example:

```
# nova quota-update --floating-ips 20 $tenant
# nova quota-show --tenant $tenant
+-----------------------------+-------+
| Property                    | Value |
+-----------------------------+-------+
| metadata_items              | 128   |
| injected_file_content_bytes | 10240 |
| ram                         | 51200 |
| floating_ips                | 20    |
| key_pairs                   | 100   |
| instances                   | 10    |
| security_group_rules        | 20    |
| injected_files              | 5     |
| cores                       | 20    |
| fixed_ips                   | -1    |
| injected_file_path_bytes    | 255   |
| security_groups             | 10    |
+-----------------------------+-------+
```

To view a list of options for the *quota-update* command, run:

```
$ nova help quota-update
```

Set Block Storage quotas

As an administrative user, you can update the Block Storage Service quotas for a tenant, as well as update the quota defaults for a new tenant.

Property Name	Description
gigabytes	Number of volume gigabytes allowed per tenant.
snapshots	Number of Block Storage snapshots allowed per tenant.
volumes	Number of Block Storage volumes allowed per tenant.

View and update Block Storage quotas for a tenant (project)

As an administrative user, you can use the *cinder quota-** commands, which are provided by the `python-cinderclient` package, to view and update tenant quotas.

To view and update default Block Storage quota values

1. List all default quotas for all tenants, as follows:

   ```
   $ cinder quota-defaults
   ```

   For example:

   ```
   $ cinder quota-defaults
   +-----------+-------+
   |  Property | Value |
   +-----------+-------+
   | gigabytes |  1000 |
   | snapshots |   10  |
   |  volumes  |   10  |
   +-----------+-------+
   ```

2. To update a default value for a new tenant, update the property in the `/etc/cinder/cinder.conf` file.

To view Block Storage quotas for a tenant

1. View quotas for the tenant, as follows:

   ```
   # cinder quota-show tenantName
   ```

   For example:

   ```
   # cinder quota-show tenant01
   +-----------+-------+
   |  Property | Value |
   +-----------+-------+
   | gigabytes |  1000 |
   | snapshots |   10  |
   |  volumes  |   10  |
   +-----------+-------+
   ```

To update Compute service quotas

1. Place the tenant ID in a useable variable, as follows:

```
$ tenant=$(keystone tenant-list | awk '/tenantName/ {print $2}')
```

2. Update a particular quota value, as follows:

```
# cinder quota-update --quotaName NewValue tenantID
```

   For example:

```
# cinder quota-update --volumes 15 $tenant
# cinder quota-show tenant01
+-----------+-------+
|  Property | Value |
+-----------+-------+
| gigabytes |  1000 |
| snapshots |   10  |
|  volumes  |   15  |
+-----------+-------+
```

User Management

The command line tools for managing users are inconvenient to use directly. They require issuing multiple commands to complete a single task, and they use UUIDs rather than symbolic names for many items. In practice, humans typically do not use these tools directly. Fortunately, the OpenStack Dashboard provides a reasonable interface to this. In addition, many sites write custom tools for local needs to enforce local policies and provide levels of self service to users that aren't currently available with packaged tools.

Creating New Users

To create a user, you need the following information:

- Username
- Email address
- Password
- Primary project
- Role

Username and email address are self-explanatory, though your site may have local conventions you should observe. Setting and changing passwords in the Identity Service requires administrative privileges. As of the Folsom release, users cannot change their own passwords. This is a large driver for creating local custom tools, and must be kept in mind when assigning and distributing passwords. The primary project is

simply the first project the user is associated with and must exist prior to creating the user. Role is almost always going to be "member". Out of the box, OpenStack comes with two roles defined:

- "member": a typical user.
- "admin": an administrative super user which has full permissions across all projects and should be used with great care.

It is possible to define other roles, but doing so is uncommon.

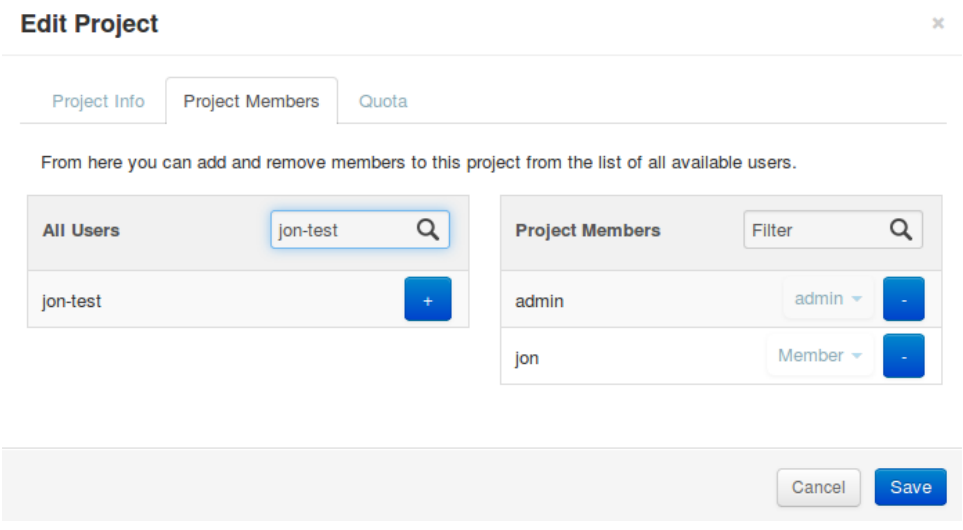
Once you've gathered this information, creating the user in the Dashboard is just another web form similar to what we've seen before and can be found on the "Users" link in the "Admin" navigation bar and then clicking the "Create User" button at the top right.

Modifying users is also done from this "Users" page. If you have a large number of users, this page can get quite crowded. The "Filter" search box at the top of the page can be used to limit the users listing. A form very similar to the user creation dialog can be pulled up by selecting "Edit" from the actions drop down menu at the end of the line for the user you are modifying.

Associating Users with Projects

Many sites run with users being associated with only one project. This is a more conservative and simpler choice both for administration and for users. Administratively if a user reports a problem with an instance or quota it is obvious which project this relates to as well. Users needn't worry about what project they are acting in if they are only in one project. However, note that, by default, any user can affect the resources of any other user within their project. It is also possible to associate users with multiple projects if that makes sense for your organization.

Associating existing users with an additional project or removing them from an older project is done from the "Projects" page of the Dashboard by selecting the "Modify Users" from the "Actions" column:



From this view you can do a number of useful and a few dangerous things.

The first column of this form, titled "All Users", will include a list of all the users in your cloud who are not already associated with this project and the second all the users who are. These can be quite long, but can be limited by typing a substring of the user name you are looking for in the filter field at the top of the column.

From here, click the + icon to add users to the project. Click the - to remove them.

The dangerous possibility comes in the ability to change member roles. This is the drop down list after the user name in the "Project Members" list. In virtually all cases this value should be set to "Member". This example purposefully show and administrative user where this value is "admin".

The "admin" is global not per project so granting a user the admin role in any project gives the administrative rights across the whole cloud.

Typical use is to only create administrative users in a single project, by convention the "admin" project which is created by default during cloud setup. If your administrative users also use the cloud to launch and manage instances it is strongly recommended that you use separate user accounts for administrative access and normal operations and that they be in distinct projects.

Customizing Authorization

The default authorization settings only allow administrative users to create resources on behalf of a different project. OpenStack handles two kind of authorization policies:

- **Operation-based**: policies specify access criteria for specific operations, possibly with fine-grained control over specific attributes.

- **Resource-based**: whether access to a specific resource might be granted or not according to the permissions configured for the resource (currently available only for the network resource). The actual authorization policies enforced in an OpenStack service vary from deployment to deployment.

The policy engine reads entries from the `policy.json` file. The actual location of this file might vary from distribution to distribution, for nova it is typically in `/etc/nova/policy.json`. You can update entries while the system is running, and you do not have to restart services. Currently the only way to update such policies is to edit the policy file.

The OpenStack service's policy engine matches a policy directly. A rule indicates evaluation of the elements of such policies. For instance, in a `compute:create: [["rule:admin_or_owner"]]` statement, the policy is `compute:create`, and the rule is `admin_or_owner`.

Policies are triggered by an OpenStack policy engine whenever one of them matches an OpenStack API operation or a specific attribute being used in a given operation. For instance, the engine tests the `create:compute` policy every time a user sends a `POST /v2/{tenant_id}/servers` request to the OpenStack Compute API server. Policies can be also related to specific API extensions. For instance, if a user needs an extension like `compute_extension:rescue` the attributes defined by the provider extensions trigger the rule test for that operation.

An authorization policy can be composed by one or more rules. If more rules are specified, evaluation policy is successful if any of the rules evaluates successfully; if an API operation matches multiple policies, then all the policies must evaluate successfully. Also, authorization rules are recursive. Once a rule is matched, the rule(s) can be resolved to another rule, until a terminal rule is reached. These are the rules defined:

- **Role-based rules**: evaluate successfully if the user submitting the request has the specified role. For instance `"role:admin"`is successful if the user submitting the request is an administrator.
- **Field-based rules:** evaluate successfully if a field of the resource specified in the current request matches a specific value. For instance `"field:networks:shared=True"` is successful if the attribute shared of the network resource is set to true.
- **Generic rules:** compare an attribute in the resource with an attribute extracted from the user's security credentials and evaluates successfully if the comparison is successful. For instance `"tenant_id:%(tenant_id)s"` is successful if the tenant

identifier in the resource is equal to the tenant identifier of the user submitting the request.

Here are snippets of the default nova `policy.json` file:

```
{
        "context_is_admin":  [["role:admin"]],
        "admin_or_owner":  [["is_admin:True"], ["project_id:%(project_id)s"]],
[1]
        "default": [["rule:admin_or_owner"]], [2]
        "compute:create": [ ],
        "compute:create:attach_network": [ ],
        "compute:create:attach_volume": [ ],
        "compute:get_all": [ ],
    "admin_api": [["is_admin:True"]],
        "compute_extension:accounts": [["rule:admin_api"]],
        "compute_extension:admin_actions": [["rule:admin_api"]],
        "compute_extension:admin_actions:pause": [["rule:admin_or_owner"]],
        "compute_extension:admin_actions:unpause": [["rule:admin_or_owner"]],
        ...
        "compute_extension:admin_actions:migrate": [["rule:admin_api"]],
        "compute_extension:aggregates": [["rule:admin_api"]],
        "compute_extension:certificates": [ ],
        ...
        "compute_extension:flavorextraspecs": [ ],
        "compute_extension:flavormanage": [["rule:admin_api"]],  [3]
        }
```

[1] Shows a rule which evaluates successfully if the current user is an administrator or the owner of the resource specified in the request (tenant identifier is equal).

[2] Shows the default policy which is always evaluated if an API operation does not match any of the policies in `policy.json`.

[3] Shows a policy restricting the ability of manipulating flavors to administrators using the Admin API only.

In some cases, some operations should be restricted to administrators only. Therefore, as a further example, let us consider how this sample policy file could be modified in a scenario where we enable users to create their own flavors:

```
"compute_extension:flavormanage": [ ],
```

Users that Disrupt Other Users

Users on your cloud can disrupt other users, sometimes intentionally and maliciously and other times by accident. Understanding the situation allows you to make a better decision on how to handle the disruption.

For example: A group of users have instances that are utilizing a large amount of compute resources for very compute-intensive tasks. This is driving the load up on compute nodes and affecting other users. In this situation, review your user use cases. You may find that high compute scenarios are common and should then plan for proper segregation in your cloud such as host aggregation or regions.

Another example is a user consuming a very large amount of bandwidth. Again, the key is to understand what the user is doing. If they naturally need a high amount of bandwidth, you might have to limit their transmission rate as to not affect other users or move them to an area with more bandwidth available. On the other hand, maybe the user's instance has been hacked and is part of a botnet launching DDOS attacks. Resolution to this issue is the same as if any other server on your network has been hacked. Contact the user and give them time to respond. If they don't respond, shut the instance down.

A final example is if a user is hammering cloud resources repeatedly. Contact the user and learn what they are trying to do. Maybe they don't understand that what they're doing is inappropriate or maybe there is an issue with the resource they are trying to access that is causing their requests to queue or lag.

One key element of systems administration that is often overlooked is that end users are the reason why systems administrators exist. Don't go the BOFH route and terminate every user who causes an alert to go off. Work with them to understand what they're trying to accomplish and see how your environment can better assist them in achieving their goals.

CHAPTER 10

User-facing Operations

This guide is for OpenStack operators and does not seek to be an exhaustive reference for users, but as an operator it is important that you have a basic understanding of how to use the cloud facilities. This chapter looks at OpenStack from a basic user perspective, which helps you understand your users' needs and determine when you get a trouble ticket whether it is a user issue or a service issue. The main concepts covered are images, flavors, security groups, blocks storage and instances.

Images

OpenStack images can often be thought of as "virtual machine templates". Images can also be standard installation mediums like ISO images. Essentially, they contain bootable file systems which are used to launch instances.

Adding Images

Several pre-made images exist and can easily be imported into the Image Service. A common image to add is the CirrOS image which is very small and used for testing purposes. To add this image, simply do:

```
# wget https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-
disk.img # glance image-create --name='cirros image' --is-public=true --
container-format=bare --disk-format=qcow2 < cirros-0.3.0-x86_64-disk.img
```

The `glance image-create` command provides a large set of options to give your image. For example, the `min-disk` option is useful for images that require root disks of a certain size (for example, large Windows images). To view these options, do:

```
$ glance help image-create
```

The `location` option is important to note. It does not copy the entire image into Glance, but reference an original location to where the image can be found. Upon launching an instance of that image, Glance accesses the image from the location specified.

The `copy-from` option copies the image from the location specified into the `/var/lib/glance/images` directory. The same thing is done when using the STDIN redirection such as shown in the example.

Run the following command to view the properties of existing images:

```
$ glance details
```

Deleting Images

To delete an image, just execute:

```
$ glance image-delete <image uuid>
```

Deleting an image does not affect instances or snapshots that were based off the image.

Other CLI Options

A full set of options can be found using:

```
$ glance help
```

or the OpenStack Image Service CLI Guide. (http://docs.openstack.org/cli/quick-start/content/glance-cli-reference.html)

The Image Service and the Database

The only thing that Glance does not store in a database is the image itself. The Glance database has two main tables:

- images
- image_properties

Working directly with the database and SQL queries can provide you with custom lists and reports of Glance images. Technically, you can update properties about images through the database, although this is not generally recommended.

Example Image Service Database Queries

One interesting example is modifying the table of images and the owner of that image. This can be easily done if you simply display the unique ID of the owner, this example goes one step further and displays the readable name of the owner:

```
$ mysql> select glance.images.id, glance.images.name, keystone.tenant.name,
is_public from glance.images inner join keystone.tenant on glance.images.own-
er=keystone.tenant.id;
```

Another example is displaying all properties for a certain image:

```
$ mysql> select name, value from image_properties where id = <image_id>
```

Flavors

Virtual hardware templates are called "flavors" in OpenStack, defining sizes for RAM, disk, number of cores and so on. The default install provides a range of five flavors. These are configurable by admin users (this too is configurable and may be delegated by redefining the access controls for `compute_extension:flavormanage` in `/etc/nova/policy.json` on the `nova-api` server). To get a list of available flavors on your system run:

```
$ nova flavor-list
```

```
+----+-----------+-----------+------+-----------+\+-------+-\+-------------+
| ID | Name      | Memory_MB | Disk | Ephemeral |/| VCPUs | /| extra_specs |
+----+-----------+-----------+------+-----------+\+-------+-\+-------------+
| 1  | m1.tiny   | 512       | 1    | 0         |/| 1     | /| {}          |
| 2  | m1.small  | 2048      | 10   | 20        |\| 1     | \| {}          |
| 3  | m1.medium | 4096      | 10   | 40        |/| 2     | /| {}          |
| 4  | m1.large  | 8192      | 10   | 80        |\| 4     | \| {}          |
| 5  | m1.xlarge | 16384     | 10   | 160       |/| 8     | /| {}          |
+----+-----------+-----------+------+-----------+\+-------+-\+-------------+
```

The `nova flavor-create` command allows authorized users to create new flavors. Additional flavor manipulation commands can be shown with the command:

```
$ nova help | grep flavor.
```

Flavors define a number of elements:

Column	Description
ID	A unique numeric id.
Name	a descriptive name. xx.size_name is conventional not required, though some third party tools may rely on it.
Memory_MB	Memory_MB: virtual machine memory in megabytes.
Disk	Virtual root disk size in gigabytes. This is an ephemeral disk the base image is copied into. When booting from a persistent volume it is not used. The "0" size is a special case which uses the native base image size as the size of the ephemeral root volume.
Ephemeral	Specifies the size of a secondary ephemeral data disk. This is an empty, unformatted disk and exists only for the life of the instance.
Swap	Optional swap space allocation for the instance.
VCPUs	Number of virtual CPUs presented to the instance.
RXTX_Factor	Optional property allows created servers to have a different bandwidth cap than that defined in the network they are attached to. This factor is multiplied by the rxtx_base property of the network. Default value is 1.0 (that is, the same as attached network).
Is_Public	Boolean value, whether flavor is available to all users or private to the tenant it was created in. Defaults to True.
extra_specs	Additional optional restrictions on which compute nodes the flavor can run on. This is implemented as key/value pairs that must match against the corresponding key/value pairs on compute nodes. Can be used to implement things like special resources (such as flavors that can only run on compute nodes with GPU hardware).

How do I modify an existing flavor?

Unfortunately, OpenStack does not provide an interface for modifying flavors, only for creating and deleting them. The OpenStack Dashboard simulates the ability to modify a flavor by deleting an existing flavor and creating a new one with the same name.

Security groups

One of the most common new user issues with OpenStack is failing to set appropriate security group when launching an instance and are then unable to contact the instance on the network.

Security groups are sets of IP filter rules that are applied to an instance's networking. They are project specific and project members can edit the default rules for their group and add new rules sets. All projects have a "default" security group which is applied to instances which have no other security group defined, unless changed this security group denies all incoming traffic.

The `nova.conf` option `allow_same_net_traffic` (which defaults to true) globally controls whether the rules applies to hosts which share a network. When set to true, hosts on the same subnet are not filtered and are allowed to pass all types of traffic between them. On a flat network, this allows all instances from all projects unfiltered communication. With VLAN networking, this allows access between instances within the same project. If `allow_same_net_traffic` is set to false, security groups are enforced for all connections, in this case it is possible for projects to simulate the `allow_same_net_traffic` by configuring their default security group to allow all traffic from their subnet.

Security groups for the current project can be found on the Horizon dashboard under "Access & Security" to see details of an existing group select the "edit" action for that security group. Obviously modifying existing groups can be done from this "edit" interface. There is a "Create Security Group" button on the main Access & Security page for creating new groups. We discuss the terms used in these fields when we explain the command line equivalents.

From the command line you can get a list of security groups for the project you're acting in using the nova command:

```
$ nova secgroup-list

+---------+-------------+
| Name    | Description |
+---------+-------------+
| default | default     |
| open    | all ports   |
+---------+-------------+
```

To view the details of the "open" security group:

```
$ nova secgroup-list-rules open
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| icmp        | -1        | 255     | 0.0.0.0/0 |              |
| tcp         | 1         | 65535   | 0.0.0.0/0 |              |
| udp         | 1         | 65535   | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

These rules are all "allow" type rules as the default is deny. The first column is the IP protocol (one of icmp, tcp, or udp) the second and third columns specify the affected port range. The third column specifies the IP range in CIDR format. This example shows the full port range for all protocols allowed from all IPs.

As noted in the previous chapter the number of rules per security group is controlled by the quota_security_group_rules and the number of allowed security groups per project is controlled by the quota_security_groups quota.

When adding a new security group you should pick a descriptive but brief name. This name shows up in brief descriptions of the instances that use it where the longer description field often does not. Seeing that an instance is using security group "http" is much easier to understand than "bobs_group" or "secgrp1".

As an example, let's create a security group that allows web traffic anywhere on the internet. We'll call this "global_http" which is clear and reasonably concise, encapsulating what is allowed and from where. From the command line:

```
+-------------+-----------------------------------+
| Name        | Description                       |
+-------------+-----------------------------------+
| global_http | allow web traffic from the internet |
+-------------+-----------------------------------+
```

This creates the empty security group to make it do what we want we need to add some rules.

```
$ nova secgroup-add-rule <secgroup> <ip-proto> <from-port> <to-port>
                                 <cidr>
$ nova secgroup-add-rule global_http tcp 80 80 0.0.0.0/0
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 80        | 80      | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

Note that the arguments are positional and the "from-port" and "to-port" arguments specify the local port range connections are allowed to not source and destination ports of the connection. More complex rule sets can be built up through multiple invocations of nova secgroup-add-rule. For example if you want to pass both http and https traffic:

```
$ nova secgroup-add-rule global_http tcp 443 443 0.0.0.0/0
```

```
+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 443       | 443     | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

Despite only outputting the newly added rule this operation is additive:

```
$ nova secgroup-list-rules global_http

+-------------+-----------+---------+-----------+--------------+
| IP Protocol | From Port | To Port | IP Range  | Source Group |
+-------------+-----------+---------+-----------+--------------+
| tcp         | 80        | 80      | 0.0.0.0/0 |              |
| tcp         | 443       | 443     | 0.0.0.0/0 |              |
+-------------+-----------+---------+-----------+--------------+
```

The inverse operation is called secgroup-delete-rule, using the same format. Whole security groups can be removed with secgroup-delete.

To create security group rules for a cluster of instances:

SourceGroups are a special dynamic way of defining the CIDR of allowed sources. The user specifies a SourceGroup (Security Group name), all the users' other Instances using the specified SourceGroup are selected dynamically. This alleviates the need for a individual rules to allow each new member of the cluster.usage:

usage: nova secgroup-add-group-rule <secgroup> <source-group> <ip-proto> <from-port> <to-port>

```
$ nova secgroup-add-group-rule cluster global-http tcp 22 22
```

The "cluster" rule allows ssh access from any other instance that uses the "global-http" group.

Block Storage

OpenStack volumes are persistent block storage devices which may be attached and detached from instances, but can only be attached to one instance at a time, similar to an external hard drive they do not proved shared storage in the way a network file system or object store does. It is left to the operating system in the instance to put a file system on the block device and mount it, or not.

Similar to other removable disk technology it is important the operating system is not trying to make use of the disk before removing it. On Linux instances this typically involves unmounting any file systems mounted from the volume. The OpenStack volume service cannot tell if it is safe to remove volumes from an instance so it does what it is told. If a user tells the volume service to detach a volume from an instance while it is being written to you can expect some level of file system corruption as well as faults from whatever process within the instance was using the device.

There is nothing OpenStack specific in being aware of the steps needed from with in the instance operating system to access block devices, potentially formatting them for first use and being cautious when removing devices. What is specific is how to create new volumes and attach and detach them from instances. These operations can all be done from the "Volumes" page of the Dashboard or using the cinder command line client.

To add new volumes you only need a name and a volume size in gigabytes, ether put these into the "create volume" web form or using the command line:

```
$ cinder create --display-name test-volume 10
```

This creates a 10 GB volume named "test-volume." To list existing volumes and the instances they are connected to if any:

```
$ cinder list

+------------+---------+--------------------+------+-------------+-------------+
|     ID     | Status  |    Display Name    | Size | Volume Type | Attached to |
+------------+---------+--------------------+------+-------------+-------------+
| 0821...19f |  active |    test-volume     |  10  |     None    |             |
+------------+---------+--------------------+------+-------------+-------------+
```

The Block Storage service also allows for creating snapshots of volumes. Remember this is a block level snapshot which is crash consistent so it is best if the volume is not connected to an instance when the snapshot is taken and second best if the volume is not in use on the instance it is attached to. If the volume is under heavy use, the snapshot may have an inconsistent file system. In fact, by default, the volume service does not take a snapshot of a volume that is attached to an image, though it can be forced. To take a volume snapshot either select "Create Snapshot" from the actions column next to the volume name in the dashboard volume page, or from the command line:

```
usage: cinder snapshot-create [--force <True|False>]
[--display-name <display-name>]
[--display-description <display-description>]
<volume-id>
Add a new snapshot.
Positional arguments:  <volume-id>           ID of the volume to snapshot
Optional arguments:  --force <True|False>  Optional flag to indicate whether to
snapshot a volume                          even if its attached to an instance.
(Default=False)  --display-name <display-name>                          Optional
snapshot name. (Default=None)
--display-description <display-description>
Optional snapshot description. (Default=None)
```

Block Storage Creation Failures

If a user tries to create a volume and it immediately goes into an error state, the best way to troubleshoot is to grep the Cinder log files for the volume's UUID. First try the

log files on the cloud controller and then try the storage node where they volume was attempted to be created:

```
# grep 903b85d0-bacc-4855-a261-10843fc2d65b /var/log/cinder/*.log
```

Instances

Instances are the running virtual machines within an OpenStack cloud. This section deals with how to work with them and their underlying images, their network properties and how they are represented in the database.

Starting Instances

To launch an instance you need to select an image, a flavor, and a name. The name needn't be unique but your life is simpler if it is because many tools will use the name in place of UUID so long as the name is unique. This can be done from the dashboard either from the "Launch Instance" button on the "Instances" page or by selecting the "Launch" action next to an image or snapshot on the "Images & Snapshots" page.

On the command line:

```
$ nova boot --flavor <flavor> --image <image> <name>
```

There are a number of optional items that can be specified. You should read the rest of this instances section before trying to start one, but this is the base command that later details are layered upon.

To delete instances from the dashboard select the "Terminate instance" action next to the instance on the "Instances" page, from the command line:

```
$ nova delete <instance-uuid>
```

It is important to note that powering off an instance does not terminate it in the OpenStack sense.

Instance Boot Failures

If an instance fails to start and immediately moves to "Error" state there are a few different ways to track down what has gone wrong. Some of these can be done with normal user access while others require access to your log server or compute nodes.

The simplest reasons for nodes to fail to launch are quota violations or the scheduler being unable to find a suitable compute node on which to run the instance. In these cases the error is apparent doing a `nova show` on the faulted instance.

```
$ nova show test-instance
```

```
+------------------------+------------------------------------------------------\
| Property               | Value                                                /
+------------------------+------------------------------------------------------\
| OS-DCF:diskConfig      | MANUAL                                               /
| OS-EXT-STS:power_state | 0                                                    \
| OS-EXT-STS:task_state  | None                                                 /
| OS-EXT-STS:vm_state    | error                                                \
| accessIPv4             |                                                      /
| accessIPv6             |                                                      \
| config_drive           |                                                      /
| created                | 2013-03-01T19:28:24Z                                 \
| fault                  | {u'message': u'NoValidHost', u'code': 500, u'created/
| flavor                 | xxl.super (11)                                       \
| hostId                 |                                                      /
| id                     | 940f3b2f-bd74-45ad-bee7-eb0a7318aa84                 \
| image                  | quantal-test (65b4f432-7375-42b6-a9b8-7f654a1e676e) /
| key_name               | None                                                 \
| metadata               | {}                                                   /
| name                   | test-instance                                        \
| security_groups        | [{u'name': u'default'}]                              /
| status                 | ERROR                                                \
| tenant_id              | 98333a1a28e746fa8c629c83a818ad57                     /
| updated                | 2013-03-01T19:28:26Z                                 \
| user_id                | a1ef823458d24a68955fec6f3d390019                     /
+------------------------+------------------------------------------------------\
```

In this case looking at the "fault" message shows NoValidHost indicating the scheduler was unable to match the instance requirements.

If `nova show` does not sufficiently explain the failure searching for the instance UUID in the `nova-compute.log` on the compute node it was scheduled on or the `nova-scheduler.log` on your scheduler hosts is a good place to start looking for lower level problems.

Using `nova show` as an admin user will show the compute node the instance was scheduled on as `hostId`, if the instance failed during scheduling this field is blank.

Instance-specific Data

There are a variety of ways to inject custom data including authorized_keys key injection, user-data, metadata service, and file injection.

To clarify user-data versus metadata, understand that "user-data" is a chunk of data, set when an instance is not running. This user-data is accessible from within the instance when it is running. People use this user-data to store configuration, a script, or anything the tenant wants.

For Compute, instance metadata is a collection of key/value pairs associated with an instance. Compute reads and writes to these key/value pairs any time during the instance lifetime, from inside and outside the instance, when the end-user uses the

Compute API to do so. However, you cannot query the instance associated key/value pairs via the metadata service that is compatible with the Amazon EC2 metadata service.

Users can generate and register ssh keys using the nova command

```
$ nova keypair-add mykey > mykey.pem
```

This creates a key named mykey which you can associate with instances. The file mykey.pem is the private key which should be saved to a secure location as it allows root access to instances the mykey key is associated with.

You can register an existing public key with OpenStack using this command

```
$ nova keypair-add --pub-key mykey.pub mykey
```

You must have the matching private key to access instances associated with this key.

To associate a key with an instance on boot add --key_name mykey to your command line for example:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --key_name mykey
```

When booting a server, you can also add metadata, so that you can more easily identify it amongst other running instances. Use the --meta option with a key=value pair, where you can make up the string for both the key and the value. For example, you could add a description and also the creator of the server.

```
$ nova boot --image=test-image --flavor=1 smallimage --meta description='Small
test image'
```

When viewing the server information, you can see the metadata included on the metadata line:

```
$ nova show smallimage

+------------------------+----------------------------------------+
|     Property           |                  Value                 |
+------------------------+----------------------------------------+
|   OS-DCF:diskConfig    |               MANUAL                   |
| OS-EXT-STS:power_state |                 1                      |
| OS-EXT-STS:task_state  |                None                    |
|  OS-EXT-STS:vm_state   |               active                   |
|    accessIPv4          |                                        |
|    accessIPv6          |                                        |
|      config_drive      |                                        |
|     created            |            2012-05-16T20:48:23Z        |
|      flavor            |              m1.small                  |
|      hostId            |             de0...487                  |
|        id              |             8ec...f915                 |
|      image             |             natty-image                |
|     key_name           |                                        |
|     metadata           | {u'description': u'Small test image'}  |
|       name             |             smallimage2                |
```

```
|    private network     |            172.16.101.11               |
|     progress           |                 0                      |
|     public network     |             10.4.113.11                |
|      status            |               ACTIVE                   |
|    tenant_id           |             e83...482                  |
|     updated            |            2012-05-16T20:48:35Z        |
|     user_id            |          de3...0a9                     |
+------------------------+----------------------------------------+
```

User Data is a special key in the metadata service which holds a file that cloud aware applications within the guest instance can access. For example cloudinit (https://help.ubuntu.com/community/CloudInit) is an open source package from Ubuntu that handles early initialization of a cloud instance that makes use of this user data.

This user-data can be put in a file on your local system and then passed in at instance creation with the flag --user-data <user-data-file> for example:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --user-data mydata.file
```

Arbitrary local files can also be placed into the instance file system at creation time using the --file <dst-path=src-path> option. You may store up to 5 files. For example if you have a special authorized_keys file named special_authorized_keysfile that you want to put on the instance rather than using the regular ssh key injection for some reason you can use the following command:

```
$ nova boot --image ubuntu-cloudimage --flavor 1 --file /root/.ssh/author-
ized_keys=special_authorized_keysfile
```

Associating Security Groups

Security groups as discussed earlier are typically required to allow network traffic to an instance, unless the default security group for a project has been modified to be more permissive.

Adding security groups is typically done on instance boot. When launching from the dashboard this is on the "Access & Security" tab of the "Launch Instance" dialog. When launching from the command line append --security-groups with a comma separated list of security groups.

It is also possible to add and remove security groups when an instance is running. Currently this is only available through the command line tools.

```
$ nova add-secgroup <server> <securitygroup>

$ nova remove-secgroup <server> <securitygroup>
```

Floating IPs

Projects have a quota controlled number of Floating IPs, however these need to be allocated by a user before they are available for use. To allocate a Floating IP to a

project there is an "Allocate IP to Project" button on the "Access & Security" page of the dashboard or on the command line by using:

```
$ nova floating-ip-create
```

Once allocated, Floating IP can be assigned to running instances from the Dashboard either by selecting the "Associate Floating IP" from the actions drop down next to the IP on the "Access & Security" page or the same action next to the instance you wish to associate it with on the "Instances" page. The inverse action, "Dissociate Floating IP", is only available from the "Access & Security" page and not from the Instances page.

From the command line, enter the following command to complete these tasks:

```
$ nova add-floating-ip <server> <address>
$ nova remove-floating-ip <server> <address>
```

Attaching Block Storage

You can attach block storage to instances from the dashboard on the Volumes page. Click the Edit Attachments action next to the volume you wish to attach.

To perform this action from command line, run the following command:

```
$ nova volume-attach <server> <volume>
```

You can also specify block device mapping at instance boot time through the nova command-line client, as follows:

```
--block-device-mapping <dev-name=mapping>
```

The block device mapping format is `<dev-name=<id>:<type>:<size(GB)>:<delete-on-terminate>`, where:

dev-name
: A device name where the volume is attached in the system at `/dev/`*`dev_name`* .

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

type
: Either `snap`, which means that the volume was created from a snapshot, or anything other than `snap` (a blank string is valid). In the example above, the volume was not created from a snapshot, so we leave this field blank in our example below.

size (GB)
: The size of the volume, in GB. It is safe to leave this blank and have the Compute service infer the size.

delete-on-terminate
: A boolean to indicate whether the volume should be deleted when the instance is terminated. True can be specified as `True` or `1`. False can be specified as `False` or `0`.

If you have previously prepared the block storage with a bootable file system image it is even possible to boot from persistent block storage. The following example will attempt boot from volume with ID=13, it does not delete on terminate. Replace the `--key-name` with a valid keypair name:

```
$ nova boot --flavor 2 --key-name mykey --block-device-mapping vda=13:::0 boot-
from-vol-test
```

Because of bug 1163566 (https://bugs.launchpad.net/nova/+bug/1163566) you must specify an image when booting from a volume in Horizon, even though this image is not used.

To boot normally from an image and attach block storage, map to a device other than vda.

Taking Snapshots

OpenStack's snapshot mechanism allows you to create new images from running instances. This is a very convenient for upgrading base images or taking a published image and customizing for local use. To snapshot a running instance to an image using the CLI:

```
$ nova image-create <instance name or uuid> <name of new image>
```

The Dashboard interface for snapshots can be confusing because the Images & Snapshots page splits content up into:

- Images
- Instance snapshots
- Volume snapshots

However, an instance snapshot *is* an image. The only difference between an image that you upload directly to glance and an image you create by snapshot is that an image created by snapshot has additional properties in the glance database. These properties are found in the image_properties table, and include:

name	value
image_type	snapshot
instance_uuid	<uuid of instance that was snapshotted>
base_image_ref	<uuid of original image of instance that was snapshotted>
image_location	snapshot

Ensuring snapshots are consistent

Content from Sébastien Han's OpenStack: Perform Consistent Snapshots blog entry (http://www.sebastien-han.fr/blog/2012/12/10/openstack-perform-consistent-snapshots/)

A snapshot captures the state of the file system, but not the state of the memory. Therefore, to ensure your snapshot contains the data that you want, before your snapshot you need to ensure that:

- Running programs have written their contents to disk
- The file system does not have any "dirty" buffers: where programs have issued the command to write to disk, but the operating system has not yet done the write

To ensure that important services have written their contents to disk (such as, databases), we recommend you read the documentation for those applications to determine what commands to issue to have them sync their contents to disk. If you are unsure how to do this, the safest approach is to simply stop these running services normally.

To deal with the "dirty" buffer issue, we recommend using the sync command before snapshotting:

```
# sync
```

Running `sync` writes dirty buffer (buffered block that have been modified but not written yet to the disk block) to disk.

Just running `sync` is not enough to ensure the file system is consistent. We recommend you use the `fsfreeze` tool, which halts new access to the file system and create a stable image on disk that is suitable for snapshotting. fsfreeze supports several file systems, including ext3, ext4, and XFS. If your virtual machine instance is running on Ubuntu, install the util-linux package to get fsfreeze:

```
# apt-get install util-linux
```

If your operating system doesn't have a version of fsfreeze available, you can use xfs_freeze instead, which is available on Ubuntu in the xfsprogs package. Despite the

"xfs" in the name, xfs_freeze also works on ext3 and ext4 if you are using a Linux kernel version 2.6.29 or greater, since it works at the virtual file system (VFS) level starting at 2.6.29. xfs_freeze supports the same command-line arguments as fsfreeze.

Consider the example where you want to take a snapshot of a persistent block storage volume, detected by the guest operating system as /dev/vdb and mounted on /mnt. The fsfreeze command accepts 2 arguments:

- -f: freeze the system
- -u: thaw (un-freeze) the system

To freeze the volume in preparation for snapshotting, you would do, as root, inside of the instance:

```
# fsfreeze -f /mnt
```

You **must mount the file system** before you run the *fsfreeze* command.

When the "fsfreeze -f" command is issued, all ongoing transactions in the file system are allowed to complete, new write system calls are halted, and other calls which modify the file system are halted. Most importantly, all dirty data, metadata, and log information are written to disk.

Once the volume has been frozen, do not attempt to read from or write to the volume, as these operations hang. The operating system stops every I/O operation and any I/O attempts is delayed until the file system has been unfrozen.

Once you have issued the fsfreeze command, it is safe to perform the snapshot. For example, if your instance was named mon-instance, and you wanted to snapshot it to an image, named mon-snapshot, you could now run the following:

```
$ nova image-create mon-instance mon-snapshot
```

When the snapshot is done, you can thaw the file system with the following command, as root, inside of the instance:

```
# fsfreeze -u /mnt
```

If you want to backup the root file system, you can't simply do the command above because it will freeze the prompt. Instead, run the following one-liner, as root, inside of the instance:

```
# fsfreeze -f / && sleep 30 && fsfreeze -u /
```

Instances in the Database

While instance information is stored in a number of database tables, the table operators are most likely to need to look at in relation to user instances is the "instances" table.

The instances table carries all most of the information related to both running and deleted instances. It has a bewildering array of fields, for an exhaustive list look at the database. These are the most useful fields for operators looking to form queries.

The “deleted” field is set to “1” if the instance has been deleted and NULL if it has not been deleted this important for excluding deleted instances from your queries.

The “uuid” field is the UUID of the instance and is used through out other tables in the database as a foreign key. This id is also reported in logs, the dashboard and command line tools to uniquely identify an instance.

A collection of foreign keys are available to find relations to the instance. The most useful of these are “user_id” and “project_id” are the UUIDs of the user who launched the instance and the project it was launched in.

The “host” field tells which compute node is hosting the instance.

The “hostname” field holds the name of the instance when it is launched. The “display-name” is initially the same as hostname but can be reset using the nova rename command.

A number of time related fields are useful for tracking when state changes happened on an instance:

- created_at
- updated_at
- deleted_at
- scheduled_at
- launched_at
- terminated_at

CHAPTER 11

Maintenance, Failures, and Debugging

Downtime, whether planned or unscheduled, is a certainty when running a cloud. This chapter aims to provide useful information for dealing proactively, or reactively with these occurrences.

Cloud Controller and Storage Proxy Failures and Maintenance

The cloud controller and storage proxy are very similar to each other when it comes to expected and unexpected downtime. One of each server type typically runs in the cloud, which makes them very noticeable when they are not running.

For the cloud controller, the good news is if your cloud is using the FlatDHCP multi-host HA network mode, existing instances and volumes continue to operate while the cloud controller is offline. However for the storage proxy, no storage traffic is possible until it is back up and running.

Planned Maintenance

One way to plan for cloud controller or storage proxy maintenance is to simply do it off-hours, such as at 1 or 2 A.M.. This strategy impacts fewer users. If your cloud controller or storage proxy is too important to have unavailable at any point in time, you must look into High Availability options.

Rebooting a cloud controller or Storage Proxy

All in all, just issue the "reboot" command. The operating system cleanly shuts services down and then automatically reboots. If you want to be very thorough, run your backup jobs just before you reboot.

After a Cloud Controller or Storage Proxy Reboots

After a cloud controller reboots, ensure that all required services were successfully started:

```
# ps aux | grep nova-
# grep AMQP /var/log/nova/nova-*.log
# ps aux | grep glance-
# ps aux | grep keystone
# ps aux | grep cinder
```

Also check that all services are functioning:

```
# source openrc
# glance index
# nova list
# keystone tenant-list
```

For the storage proxy, ensure that the Object Storage service has resumed:

```
# ps aux | grep swift
```

Also check that it is functioning:

```
# swift stat
```

Total Cloud Controller Failure

Unfortunately, this is a rough situation. The cloud controller is a integral part of your cloud. If you have only one controller, many services are missing.

To avoid this situation, create a highly available cloud controller cluster. This is outside the scope of this document, but you can read more in the draft OpenStack High Availability Guide (http://docs.openstack.org/trunk/openstack-ha/content/ch-intro.html).

The next best way is to use a configuration management tool such as Puppet to automatically build a cloud controller. This should not take more than 15 minutes if you have a spare server available. After the controller rebuilds, restore any backups taken (see the **Backup and Recovery** chapter).

Also, in practice, sometimes the nova-compute services on the compute nodes do not reconnect cleanly to rabbitmq hosted on the controller when it comes back up after a long reboot and a restart on the nova services on the compute nodes is required.

Compute Node Failures and Maintenance

Sometimes a compute node either crashes unexpectedly or requires a reboot for maintenance reasons.

Planned Maintenance

If you need to reboot a compute node due to planned maintenance (such as a software or hardware upgrade), first ensure that all hosted instances have been moved off of the node. If your cloud is utilizing shared storage, use the `nova live-migration` command. First, get a list of instances that need to be moved:

```
# nova list --host c01.example.com --all-tenants
```

Next, migrate them one by one:

```
# nova live-migration <uuid> c02.example.com
```

If you are not using shared storage, you can use the `--block-migrate` option:

```
# nova live-migration --block-migrate <uuid> c02.example.com
```

After you have migrated all instances, ensure the `nova-compute` service has stopped:

```
# stop nova-compute
```

If you use a configuration management system, such as Puppet, that ensures the `nova-compute` service is always running, you can temporarily move the init files:

```
# mkdir /root/tmp
# mv /etc/init/nova-compute.conf /root/tmp
# mv /etc/init.d/nova-compute /root/tmp
```

Next, shut your compute node down, perform your maintenance, and turn the node back on. You can re-enable the `nova-compute` service by undoing the previous commands:

```
# mv /root/tmp/nova-compute.conf /etc/init
# mv /root/tmp/nova-compute /etc/init.d/
```

Then start the `nova-compute` service:

```
# start nova-compute
```

You can now optionally migrate the instances back to their original compute node.

After a Compute Node Reboots

When you reboot a compute node, first verify that it booted successfully. This includes ensuring the `nova-compute` service is running:

```
# ps aux | grep nova-compute
# status nova-compute
```

Also ensure that it has successfully connected to the AMQP server:

```
# grep AMQP /var/log/nova/nova-compute
2013-02-26 09:51:31 12427 INFO nova.openstack.common.rpc.common [-] Connected
to AMQP server on 199.116.232.36:5672
```

After the compute node is successfully running, you must deal with the instances that are hosted on that compute node as none of them is running. Depending on your SLA with your users or customers, you might have to start each instance and ensure they start correctly.

Instances

You can create a list of instances that are hosted on the compute node by performing the following command:

```
# nova list --host c01.example.com --all-tenants
```

After you have the list, you can use the nova command to start each instance:

```
# nova reboot <uuid>
```

Any time an instance shuts down unexpectedly, it might have problems on boot. For example, the instance might require an `fsck` on the root partition. If this happens, the user can use the Dashboard VNC console to fix this.

If an instance does not boot, meaning `virsh list` never shows the instance as even attempting to boot, do the following on the compute node:

```
# tail -f /var/log/nova/nova-compute.log
```

Try executing the `nova reboot` command again. You should see an error message about why the instance was not able to boot

In most cases, the error is due to something in libvirt's XML file (`/etc/libvirt/qemu/instance-xxxxxxxx.xml`) that no longer exists. You can enforce recreation of the XML file as well as rebooting the instance by running:

```
# nova reboot --hard <uuid>
```

Inspecting and Recovering Data from Failed Instances

In some scenarios, instances are running but are inaccessible through SSH and do not respond to any command. VNC console could be displaying a boot failure or kernel panic error messages. This could be an indication of a file system corruption on the VM itself. If you need to recover files or inspect the content of the instance, qemu-nbd can be used to mount the disk.

If you access or view the user's content and data, get their approval first!

To access the instance's disk (/var/lib/nova/instances/instance-xxxxxx/disk), the following steps must be followed:

1. Suspend the instance using the virsh command
2. Connect the qemu-nbd device to the disk
3. Mount the qemu-nbd device
4. Unmount the device after inspecting
5. Disconnect the qemu-nbd device
6. Resume the instance

If you do not follow the steps from 4-6, OpenStack Compute cannot manage the instance any longer. It fails to respond to any command issued by OpenStack Compute and it is marked as shutdown.

Once you mount the disk file, you should be able access it and treat it as normal directories with files and a directory structure. However, we do not recommend that you edit or touch any files because this could change the acls and make the instance unbootable if it is not already.

1. Suspend the instance using the virsh command - taking note of the internal ID.

```
root@compute-node:~# virsh list
Id Name                 State
----------------------------------
1 instance-00000981    running
2 instance-000009f5    running
30 instance-0000274a    running

root@compute-node:~# virsh suspend 30
Domain 30 suspended
```

2. Connect the qemu-nbd device to the disk

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# ls -lh
total 33M
-rw-rw---- 1 libvirt-qemu kvm  6.3K Oct 15 11:31 console.log
-rw-r--r-- 1 libvirt-qemu kvm   33M Oct 15 22:06 disk
-rw-r--r-- 1 libvirt-qemu kvm  384K Oct 15 22:06 disk.local
-rw-rw-r-- 1 nova         nova 1.7K Oct 15 11:30 libvirt.xml
root@compute-node:/var/lib/nova/instances/instance-0000274a#    qemu-nbd    -
c /dev/nbd0 `pwd`/disk
```

3. Mount the qemu-nbd device.

   The qemu-nbd device tries to export the instance disk's different partitions as separate devices. For example if vda as the disk and vda1 as the root partition, qemu-nbd exports the device as /dev/nbd0 and /dev/nbd0p1 respectively.

```
#mount the root partition of the device
root@compute-node:/var/lib/nova/instances/instance-0000274a#   mount   /dev/
nbd0p1 /mnt/
# List the directories of mnt, and the vm's folder is display
# You can inspect the folders and access the /var/log/ files
```

To examine the secondary or ephemeral disk, use an alternate mount point if you want both primary and secondary drives mounted at the same time.

```
# umount /mnt
# qemu-nbd -c /dev/nbd1 `pwd`/disk.local
# mount /dev/nbd1 /mnt/

root@compute-node:/var/lib/nova/instances/instance-0000274a# ls -lh /mnt/
total 76K
lrwxrwxrwx.  1 root root    7 Oct 15 00:44 bin -> usr/bin
dr-xr-xr-x.  4 root root 4.0K Oct 15 01:07 boot
drwxr-xr-x.  2 root root 4.0K Oct 15 00:42 dev
drwxr-xr-x. 70 root root 4.0K Oct 15 11:31 etc
drwxr-xr-x.  3 root root 4.0K Oct 15 01:07 home
lrwxrwxrwx.  1 root root    7 Oct 15 00:44 lib -> usr/lib
lrwxrwxrwx.  1 root root    9 Oct 15 00:44 lib64 -> usr/lib64
drwx------.  2 root root  16K Oct 15 00:42 lost+found
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 media
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 mnt
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 opt
drwxr-xr-x.  2 root root 4.0K Oct 15 00:42 proc
dr-xr-x---.  3 root root 4.0K Oct 15 21:56 root
drwxr-xr-x. 14 root root 4.0K Oct 15 01:07 run
lrwxrwxrwx.  1 root root    8 Oct 15 00:44 sbin -> usr/sbin
drwxr-xr-x.  2 root root 4.0K Feb  3  2012 srv
drwxr-xr-x.  2 root root 4.0K Oct 15 00:42 sys
drwxrwxrwt.  9 root root 4.0K Oct 15 16:29 tmp
drwxr-xr-x. 13 root root 4.0K Oct 15 00:44 usr
drwxr-xr-x. 17 root root 4.0K Oct 15 00:44 var
```

4. Once you have completed the inspection, umount the mount point and release the qemu-nbd device

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# umount /mnt
root@compute-node:/var/lib/nova/instances/instance-0000274a#    qemu-nbd    -
d /dev/nbd0
/dev/nbd0 disconnected
```

5. Resume the instance using virsh

```
root@compute-node:/var/lib/nova/instances/instance-0000274a# virsh list
Id Name                 State
----------------------------------
1 instance-00000981    running
2 instance-000009f5    running
30 instance-0000274a    paused

root@compute-node:/var/lib/nova/instances/instance-0000274a# virsh resume 30
Domain 30 resumed
```

Volumes

If the affected instances also had attached volumes, first generate a list of instance and volume UUIDs:

```
mysql> select nova.instances.uuid as instance_uuid, cinder.volumes.id as vol-
ume_uuid, cinder.volumes.status,
cinder.volumes.attach_status,   cinder.volumes.mountpoint,   cinder.volumes.dis-
play_name from cinder.volumes
inner join nova.instances on cinder.volumes.instance_uuid=nova.instances.uuid
 where nova.instances.host = 'c01.example.com';
```

You should see a result like the following:

```
+--------------+------------+-------+--------------+-----------+--------------+
|instance_uuid |volume_uuid |status |attach_status |mountpoint | display_name |
+--------------+------------+-------+--------------+-----------+--------------+
|9b969a05      |1f0fbf36    |in-use |attached      |/dev/vdc   | test         |
+--------------+------------+-------+--------------+-----------+--------------+
1 row in set (0.00 sec)
```

Next, manually detach and reattach the volumes:

```
# nova volume-detach <instance_uuid> <volume_uuid>
# nova volume-attach <instance_uuid> <volume_uuid> /dev/vdX
```

Where X is the proper mount point. Make sure that the instance has successfully booted and is at a login screen before doing the above.

Total Compute Node Failure

If a compute node fails and won't be fixed for a few hours or ever, you can relaunch all instances that are hosted on the failed node if you use shared storage for `/var/lib/nova/instances`.

To do this, generate a list of instance UUIDs that are hosted on the failed node by running the following query on the nova database:

```
mysql> select uuid from instances where host = 'c01.example.com' and deleted =
0;
```

Next, tell Nova that all instances that used to be hosted on c01.example.com are now hosted on c02.example.com:

```
mysql> update instances set host = 'c02.example.com' where host = 'c01.exam-
ple.com' and deleted = 0;
```

After that, use the nova command to reboot all instances that were on c01.example.com while regenerating their XML files at the same time:

```
# nova reboot --hard <uuid>
```

Finally, re-attach volumes using the same method described in **Volumes**.

/var/lib/nova/instances

It's worth mentioning this directory in the context of failed compute nodes. This directory contains the libvirt KVM file-based disk images for the instances that are hosted on that compute node. If you are not running your cloud in a shared storage environment, this directory is unique across all compute nodes.

`/var/lib/nova/instances` contains two types of directories.

The first is the `_base` directory. This contains all of the cached base images from glance for each unique image that has been launched on that compute node. Files ending in `_20` (or a different number) are the ephemeral base images.

The other directories are titled `instance-xxxxxxxx`. These directories correspond to instances running on that compute node. The files inside are related to one of the files in the `_base` directory. They're essentially differential-based files containing only the changes made from the original `_base` directory.

All files and directories in `/var/lib/nova/instances` are uniquely named. The files in _base are uniquely titled for the glance image that they are based on and the directory names `instance-xxxxxxxx` are uniquely titled for that particular instance. For example, if you copy all data from `/var/lib/nova/instances` on one compute node to another, you do not overwrite any files or cause any damage to images that have the same unique name, because they are essentially the same file.

Although this method is not documented or supported, you can use it when your compute node is permanently offline but you have instances locally stored on it.

Storage Node Failures and Maintenance

Due to the Object Storage's high redundancy, dealing with object storage node issues is a lot easier than dealing with compute node issues.

Rebooting a Storage Node

If a storage node requires a reboot, simply reboot it. Requests for data hosted on that node are redirected to other copies while the server is rebooting.

Shutting Down a Storage Node

If you need to shut down a storage node for an extended period of time (1+ days), consider removing the node from the storage ring. For example:

```
# swift-ring-builder account.builder remove <ip address of storage node>
# swift-ring-builder container.builder remove <ip address of storage node>
# swift-ring-builder object.builder remove <ip address of storage node>
# swift-ring-builder account.builder rebalance
# swift-ring-builder container.builder rebalance
# swift-ring-builder object.builder rebalance
```

Next, redistribute the ring files to the other nodes:

```
# for i in s01.example.com s02.example.com s03.example.com
> do
> scp *.ring.gz $i:/etc/swift
> done
```

These actions effectively take the storage node out of the storage cluster.

When the node is able to rejoin the cluster, just add it back to the ring. The exact syntax to add a node to your Swift cluster using `swift-ring-builder` heavily depends on the original options used when you originally created your cluster. Please refer back to those commands.

Replacing a Swift Disk

If a hard drive fails in a Object Storage node, replacing it is relatively easy. This assumes that your Object Storage environment is configured correctly where the data that is stored on the failed drive is also replicated to other drives in the Object Storage environment.

This example assumes that `/dev/sdb` has failed.

First, unmount the disk:

```
# umount /dev/sdb
```

Next, physically remove the disk from the server and replace it with a working disk.

Ensure that the operating system has recognized the new disk:

```
# dmesg | tail
```

You should see a message about /dev/sdb.

Because it is recommended to not use partitions on a swift disk, simply format the disk as a whole:

```
# mkfs.xfs /dev/sdb
```

Finally, mount the disk:

```
# mount -a
```

Swift should notice the new disk and that no data exists. It then begins replicating the data to the disk from the other existing replicas.

Handling a Complete Failure

A common way of dealing with the recovery from a full system failure, such as a power outage of a data center is to assign each service a priority, and restore in order.

1	Internal network connectivity
2	Backing storage services
3	Public network connectivity for user Virtual Machines
4	Nova-compute, nova-network, cinder hosts
5	User virtual machines
10	Message Queue and Database services
15	Keystone services
20	cinder-scheduler
21	Image Catalogue and Delivery services
22	nova-scheduler services
98	Cinder-api
99	Nova-api services
100	Dashboard node

Use this example priority list to ensure that user affected services are restored as soon as possible, but not before a stable environment is in place. Of course, despite being

listed as a single line item, each step requires significant work. For example, just after starting the database, you should check its integrity or, after starting the Nova services, you should verify that the hypervisor matches the database and fix any mismatches.

Configuration Management

Maintaining an OpenStack cloud requires that you manage multiple physical servers, and this number might grow over time. Because managing nodes manually is error-prone, we strongly recommend that you use a configuration management tool. These tools automate the process of ensuring that all of your nodes are configured properly and encourage you to maintain your configuration information (such as packages and configuration options) in a version controlled repository.

Several configuration management tools are available, and this guide does not recommend a specific one. The two most popular ones in the OpenStack community are Puppet (https://puppetlabs.com/) with available OpenStack Puppet modules (http://github.com/puppetlabs/puppetlabs-openstack) and Chef (http://opscode.com/chef) with available OpenStack Chef recipes (https://github.com/opscode/openstack-chef-repo). Other newer configuration tools include Juju (https://juju.ubuntu.com/) Ansible (http://ansible.cc) and Salt (http://saltstack.com), and more mature configuration management tools include CFEngine (http://cfengine.com) and Bcfg2 (http://bcfg2.org).

Working with Hardware

Similar to your initial deployment, you should ensure all hardware is appropriately burned in before adding it to production. Run software that uses the hardware to its limits - maxing out RAM, CPU, disk and network. Many options are available, and normally double as benchmark software so you also get a good idea of the performance of your system.

Adding a Compute Node

If you find that you have reached or are reaching the capacity limit of your computing resources, you should plan to add additional compute nodes. Adding more nodes is quite easy. The process for adding nodes is the same as when the initial compute nodes were deployed to your cloud: use an automated deployment system to bootstrap the bare-metal server with the operating system and then have a configuration management system install and configure the OpenStack Compute service. Once the Compute service has been installed and configured in the same way as the other compute nodes, it automatically attaches itself to the cloud. The cloud controller notices the new node(s) and begin scheduling instances to launch there.

If your OpenStack Block Storage nodes are separate from your compute nodes, the same procedure still applies as the same queuing and polling system is used in both services.

We recommend that you use the same hardware for new compute and block storage nodes. At the very least, ensure that the CPUs are similar in the compute nodes to not break live migration.

Adding an Object Storage Node

Adding a new object storage node is different than adding compute or block storage nodes. You still want to initially configure the server by using your automated deployment and configuration management systems. After that is done, you need to add the local disks of the object storage node into the object storage ring. The exact command to do this is the same command that was used to add the initial disks to the ring. Simply re-run this command on the object storage proxy server for all disks on the new object storage node. Once this has been done, rebalance the ring and copy the resulting ring files to the other storage nodes.

If your new object storage node has a different number of disks than the original nodes have, the command to add the new node is different than the original commands. These parameters vary from environment to environment.

Replacing Components

Failures of hardware are common in large scale deployments such as an infrastructure cloud. Consider your processes and balance time saving against availability. For example, an Object Storage cluster can easily live with dead disks in it for some period of time if it has sufficient capacity. Or, if your compute installation is not full you could consider live migrating instances off a host with a RAM failure until you have time to deal with the problem.

Databases

Almost all OpenStack components have an underlying database to store persistent information. Usually this database is MySQL. Normal MySQL administration is applicable to these databases. OpenStack does not configure the databases out of the ordinary. Basic administration includes performance tweaking, high availability, backup, recovery, and repairing. For more information, see a standard MySQL administration guide.

You can perform a couple tricks with the database to either more quickly retrieve information or fix a data inconsistency error. For example, an instance was terminated

but the status was not updated in the database. These tricks are discussed throughout this book.

Database Connectivity

Review the components configuration file to see how each OpenStack component accesses its corresponding database. Look for either `sql_connection` or simply `connection`:

```
# grep -hE "connection ?=" /etc/nova/nova.conf /etc/glance/glance-*.conf
/etc/cinder/cinder.conf /etc/keystone/keystone.conf
        sql_connection = mysql://nova:nova@cloud.alberta.sandbox.cybera.ca/nova
        sql_connection = mysql://glance:password@cloud.example.com/glance
        sql_connection = mysql://glance:password@cloud.example.com/glance
    sql_connection=mysql://cinder:password@cloud.example.com/cinder
        connection = mysql://keystone_admin:password@cloud.example.com/keystone
```

The connection strings take this format:

```
mysql:// <username> : <password> @ <hostname> / <database name>
```

Performance and Optimizing

As your cloud grows, MySQL is utilized more and more. If you suspect that MySQL might be becoming a bottleneck, you should start researching MySQL optimization. The MySQL manual has an entire section dedicated to this topic Optimization Overview (http://dev.mysql.com/doc/refman/5.5/en/optimize-overview.html).

HDWMY

Here's a quick list of various to-do items each hour, day, week, month, and year. Please note these tasks are neither required nor definitive, but helpful ideas:

Hourly

- Check your monitoring system for alerts and act on them.
- Check your ticket queue for new tickets.

Daily

- Check for instances in a failed or weird state and investigate why.
- Check for security patches and apply them as needed.

Weekly

- Check cloud usage:
  - User quotas
  - Disk space
  - Image usage
  - Large instances
  - Network usage (bandwidth and IP usage)
- Verify your alert mechanisms are still working.

Monthly

- Check usage and trends over the past month.
- Check for user accounts that should be removed.
- Check for operator accounts that should be removed.

Quarterly

- Review usage and trends over the past quarter.
- Prepare any quarterly reports on usage and statistics.
- Review and plan any necessary cloud additions.
- Review and plan any major OpenStack upgrades.

Semi-Annually

- Upgrade OpenStack.
- Clean up after OpenStack upgrade (any unused or new services to be aware of?)

Determining which Component Is Broken

OpenStack's collection of different components interact with each other strongly. For example, uploading an image requires interaction from `nova-api`, `glance-api`, `glance-registry`, Keystone, and potentially `swift-proxy`. As a result, it is sometimes difficult to determine exactly where problems lie. Assisting in this is the purpose of this section.

Tailing Logs

The first place to look is the log file related to the command you are trying to run. For example, if `nova list` is failing, try tailing a Nova log file and running the command again:

Terminal 1:

```
# tail -f /var/log/nova/nova-api.log
```

Terminal 2:

```
# nova list
```

Look for any errors or traces in the log file. For more information, see the chapter on **Logging and Monitoring**.

If the error indicates that the problem is with another component, switch to tailing that component's log file. For example, if nova cannot access glance, look at the glance-api log:

Terminal 1:

```
# tail -f /var/log/glance/api.log
```

Terminal 2:

```
# nova list
```

Wash, rinse, repeat until you find the core cause of the problem.

Running Daemons on the CLI

Unfortunately, sometimes the error is not apparent from the log files. In this case, switch tactics and use a different command, maybe run the service directly on the command line. For example, if the `glance-api` service refuses to start and stay running, try launching the daemon from the command line:

```
# sudo -u glance -H glance-api
```

This might print the error and cause of the problem.

The `-H` flag is required when running the daemons with sudo because some daemons will write files relative to the user's home directory, and this write may fail if `-H` is left off.

Example of Complexity

One morning, a compute node failed to run any instances. The log files were a bit vague, claiming that a certain instance was unable to be started. This ended up being

a red herring because the instance was simply the first instance in alphabetical order, so it was the first instance that nova-compute would touch.

Further troubleshooting showed that libvirt was not running at all. This made more sense. If libvirt wasn't running, then no instance could be virtualized through KVM. Upon trying to start libvirt, it would silently die immediately. The libvirt logs did not explain why.

Next, the `libvirtd` daemon was run on the command line. Finally a helpful error message: it could not connect to d-bus. As ridiculous as it sounds, libvirt, and thus `nova-compute`, relies on d-bus and somehow d-bus crashed. Simply starting d-bus set the entire chain back on track and soon everything was back up and running.

Upgrades

With the exception of Object Storage, an upgrade from one version of OpenStack to another is a great deal of work.

The upgrade process generally follows these steps:

1. Read the release notes and documentation.
2. Find incompatibilities between different versions.
3. Plan an upgrade schedule and complete it in order on a test cluster.
4. Run the upgrade.

You can perform an upgrade while user instances run. However, this strategy can be dangerous. Don't forget appropriate notice to your users, and backups.

The general order that seems to be most successful is:

1. Upgrade the OpenStack Identity service (keystone).
2. Upgrade the OpenStack Image service (glance).
3. Upgrade all OpenStack Compute (nova) services.
4. Upgrade all OpenStack Block Storage (cinder) services.

For each of these steps, complete the following sub-steps:

1. Stop services.
2. Create a backup of configuration files and databases.
3. Upgrade the packages using your distribution's package manager.
4. Update the configuration files according to the release notes.
5. Apply the database upgrades.

6. Restart the services.
7. Verify that everything is running.

Probably the most important step of all is the pre-upgrade testing. Especially if you are upgrading immediately after release of a new version, undiscovered bugs might hinder your progress. Some deployers prefer to wait until the first point release is announced. However, if you have a significant deployment, you might follow the development and testing of the release, thereby ensuring that bugs for your use cases are fixed.

To complete an upgrade of OpenStack Compute while keeping instances running, you should be able to use live migration to move machines around while performing updates, and then move them back afterward as this is a property of the hypervisor. However, it is critical to ensure that database changes are successful otherwise an inconsistent cluster state could arise.

Performing some 'cleaning' of the cluster prior to starting the upgrade is also a good idea, to ensure the state is consistent. For example some have reported issues with instances that were not fully removed from the system after their deletion. Running a command equivalent to:

```
$ virsh list --all
```

to find deleted instances that are still registered in the hypervisor and removing them prior to running the upgrade can avoid issues.

Uninstalling

While we'd always recommend using your automated deployment system to re-install systems from scratch, sometimes you do need to remove OpenStack from a system the hard way. Here's how:

- Remove all packages
- Remove remaining files
- Remove databases

These steps depend on your underlying distribution, but in general you should be looking for 'purge' commands in your package manager, like `aptitude purge ~c $package`. Following this, you can look for orphaned files in the directories referenced throughout this guide. For uninstalling the database properly, refer to the manual appropriate for the product in use.

CHAPTER 12

Network Troubleshooting

Network troubleshooting can unfortunately be a very difficult and confusing procedure. A network issue can cause a problem at several points in the cloud. Using a logical troubleshooting procedure can help mitigate the confusion and more quickly isolate where exactly the network issue is. This chapter aims to give you the information you need to make yours.

Using "ip a" to Check Interface States

On compute nodes and nodes running nova-network, use the following command to see information about interfaces, including information about IPs, VLANs, and whether your interfaces are up.

```
# ip a
```

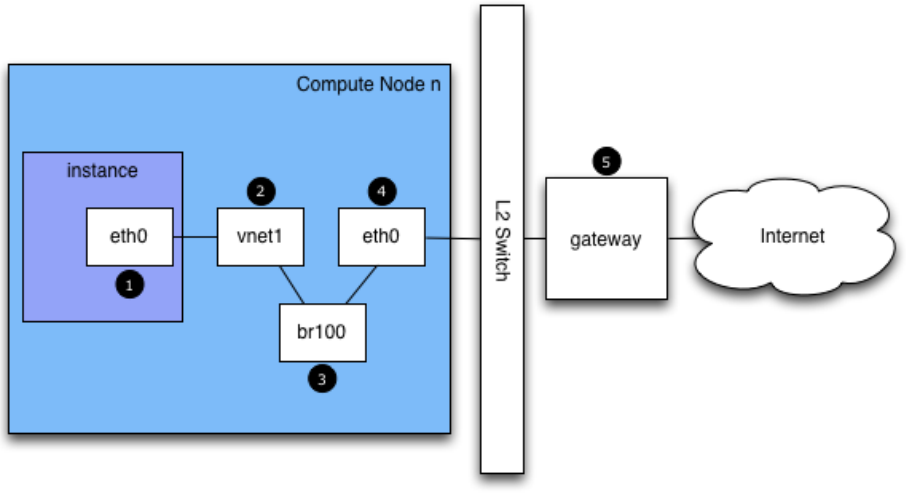
If you're encountering any sort of networking difficulty, one good initial sanity check is to make sure that your interfaces are up. For example:

```
$ ip a | grep state
1: lo: <LOOPBACK,UP,LOWER_UP> mtu 16436 qdisc noqueue state UNKNOWN
2: eth0: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP
qlen 1000
3: eth1: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast master
br100 state UP qlen 1000
4: virbr0: <NO-CARRIER,BROADCAST,MULTICAST,UP> mtu 1500 qdisc noqueue state
DOWN
6: br100: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue state UP
```

You can safely ignore the state of virbr0, which is a default bridge created by libvirt and not used by OpenStack.

Network Traffic in the Cloud

If you are logged in to an instance and ping an external host, for example google.com, the ping packet takes the following route:



1. The instance generates a packet and places it on the virtual NIC inside the instance, such as, eth0.
2. The packet transfers to the virtual NIC of the compute host, such as, vnet1. You can find out what vent NIC is being used by looking at the /etc/libvirt/qemu/instance-xxxxxxxx.xml file.
3. From the vnet NIC, the packet transfers to a bridge on the compute node, such as, `br100`.

   If you run FlatDHCPManager, one bridge is on the compute node. If you run VlanManager, one bridge exists for each VLAN.

   To see which bridge the packet will use, run the command:

   ```
   $ brctl show
   ```

   Look for the vnet NIC. You can also reference nova.conf and look for the flat_interface_bridge option.
4. The packet transfers to the main NIC of the compute node. You can also see this NIC in the brctl output, or you can find it by referencing the flat_interface option in nova.conf.
5. After the packet is on this NIC, it transfers to the compute node's default gateway. The packet is now most likely out of your control at this point. The diagram de-

picts an external gateway. However, in the default configuration with multi-host, the compute host is the gateway.

Reverse the direction to see the path of a ping reply.

From this path, you can see that a single packet travels across four different NICs. If a problem occurs with any of these NICs, a network issue occurs.

Finding a Failure in the Path

Use ping to quickly find where a failure exists in the network path. In an instance, first see if you can ping an external host, such as google.com. If you can, then there shouldn't be a network problem at all.

If you can't, try pinging the IP address of the compute node where the instance is hosted. If you can ping this IP, then the problem is somewhere between the compute node and that compute node's gateway.

If you can't ping the IP address of the compute node, the problem is between the instance and the compute node. This includes the bridge connecting the compute node's main NIC with the vnet NIC of the instance.

One last test is to launch a second instance and see if the two instances can ping each other. If they can, the issue might be related to the firewall on the compute node.

tcpdump

One great, although very in-depth, way of troubleshooting network issues is to use tcpdump. It's recommended to use tcpdump at several points along the network path to correlate where a problem might be. If you prefer working with a GUI, either live or by using a tcpdump capture do also check out Wireshark (http://www.wireshark.org/).

For example, run the following command:

```
tcpdump -i any -n -v 'icmp[icmptype] = icmp-echoreply or icmp[icmptype]
= icmp-echo'
```

Run this on the command line of the following areas:

1. An external server outside of the cloud.
2. A compute node.
3. An instance running on that compute node.

In this example, these locations have the following IP addresses:

```
Instance
10.0.2.24
203.0.113.30
Compute Node
10.0.0.42
203.0.113.34
External Server
1.2.3.4
```

Next, open a new shell to the instance and then ping the external host where tcpdump is running. If the network path to the external server and back is fully functional, you see something like the following:

On the external server:

```
12:51:42.020227 IP (tos 0x0, ttl 61, id 0, offset 0, flags [DF], proto ICMP
(1), length 84)
    203.0.113.30 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.020255 IP (tos 0x0, ttl 64, id 8137, offset 0, flags [none], proto
ICMP (1), length 84)
    1.2.3.4 > 203.0.113.30: ICMP echo reply, id 24895, seq 1, length 64
```

On the Compute Node:

```
12:51:42.019519 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP
(1), length 84)
    10.0.2.24 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.019519 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto ICMP
(1), length 84)
    10.0.2.24 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.019545 IP (tos 0x0, ttl 63, id 0, offset 0, flags [DF], proto ICMP
(1), length 84)
    203.0.113.30 > 1.2.3.4: ICMP echo request, id 24895, seq 1, length 64
12:51:42.019780 IP (tos 0x0, ttl 62, id 8137, offset 0, flags [none], proto
ICMP (1), length 84)
    1.2.3.4 > 203.0.113.30: ICMP echo reply, id 24895, seq 1, length 64
12:51:42.019801 IP (tos 0x0, ttl 61, id 8137, offset 0, flags [none], proto
ICMP (1), length 84)
    1.2.3.4 > 10.0.2.24: ICMP echo reply, id 24895, seq 1, length 64
12:51:42.019807 IP (tos 0x0, ttl 61, id 8137, offset 0, flags [none], proto
ICMP (1), length 84)
    1.2.3.4 > 10.0.2.24: ICMP echo reply, id 24895, seq 1, length 64
```

On the Instance:

```
12:51:42.020974 IP (tos 0x0, ttl 61, id 8137, offset 0, flags [none], proto
ICMP (1), length 84)
 1.2.3.4 > 10.0.2.24: ICMP echo reply, id 24895, seq 1, length 64
```

Here, the external server received the ping request and sent a ping reply. On the compute node, you can see that both the ping and ping reply successfully passed through.

You might also see duplicate packets on the compute node, as seen above, because tcpdump captured the packet on both the bridge and outgoing interface.

iptables

Nova automatically manages iptables, including forwarding packets to and from instances on a compute node, forwarding floating IP traffic, and managing security group rules.

Run the following command to view the current iptables configuration:

```
# iptables-save
```

If you modify the configuration, it reverts the next time you restart nova-network. You must use OpenStack to manage iptables.

Network Configuration in the Database

The nova database table contains a few tables with networking information:

- fixed_ips: contains each possible IP address for the subnet(s) added to Nova. This table is related to the instances table by way of the fixed_ips.instance_uuid column.
- floating_ips: contains each floating IP address that was added to nova. This table is related to the fixed_ips table by way of the floating_ips.fixed_ip_id column.
- instances: not entirely network specific, but it contains information about the instance that is utilizing the fixed_ip and optional floating_ip.

From these tables, you can see that a Floating IP is technically never directly related to an instance, it must always go through a Fixed IP.

Manually De-Associating a Floating IP

Sometimes an instance is terminated but the Floating IP was not correctly de-associated from that instance. Because the database is in an inconsistent state, the usual tools to de-associate the IP no longer work. To fix this, you must manually update the database.

First, find the UUID of the instance in question:

```
mysql> select uuid from instances where hostname = 'hostname';
```

Next, find the Fixed IP entry for that UUID:

```
mysql> select * from fixed_ips where instance_uuid = '<uuid>';
```

You can now get the related Floating IP entry:

```
mysql> select * from floating_ips where fixed_ip_id = '<fixed_ip_id>';
```

And finally, you can de-associate the Floating IP:

```
mysql> update floating_ips set fixed_ip_id = NULL, host = NULL where
fixed_ip_id = '<fixed_ip_id>';
```

You can optionally also de-allocate the IP from the user's pool:

```
mysql> update floating_ips set project_id = NULL where fixed_ip_id =
'<fixed_ip_id>';
```

Debugging DHCP Issues

One common networking problem is that an instance boots successfully but is not reachable because it failed to obtain an IP address from dnsmasq, which is the DHCP server that is launched by the nova-network service.

The simplest way to identify that this the problem with your instance is to look at the console output of your instance. If DHCP failed, you can retrieve the console log by doing:

```
$ nova console-log <instance name or uuid>
```

If your instance failed to obtain an IP through DHCP, some messages should appear in the console. For example, for the Cirros image, you see output that looks like:

```
udhcpc (v1.17.2) started
Sending discover...
Sending discover...
Sending discover...
No lease, forking to background
starting DHCP forEthernet interface eth0 [ [1;32mOK[0;39m ]
cloud-setup: checking http://169.254.169.254/2009-04-04/meta-data/instance-id
wget: can't connect to remote host (169.254.169.254): Network is unreachable
```

After you establish that the instance booted properly, the task is to figure out where the failure is.

A DHCP problem might be caused by a misbehaving dnsmasq process. First, debug by checking logs and then restart the dnsmasq processes only for that project (tenant). In VLAN mode there is a dnsmasq process for each tenant. Once you have restarted targeted dnsmasq processes, the simplest way to rule out dnsmasq causes is to kill all of the dnsmasq processes on the machine, and restart nova-network. As a last resort, do this as root:

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

It's openstack-nova-network on RHEL/CentOS/Fedora but nova-network on Ubuntu/Debian.

Several minutes after nova-network is restarted, you should see new dnsmasq processes running:

```
# ps aux | grep dnsmasq
nobody 3735 0.0 0.0 27540 1044 ? S 15:40 0:00 /usr/sbin/dnsmasq --strict-order
--bind-interfaces --conf-file=
        --domain=novalocal  --pid-file=/var/lib/nova/networks/nova-br100.pid  --
listen-address=192.168.100.1
      --except-interface=lo  --dhcp-range=set:'novanetwork',192.168.100.2,static,
120s --dhcp-lease-max=256
              --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf    --dhcp-
script=/usr/bin/nova-dhcpbridge --leasefile-ro
root 3736 0.0 0.0 27512 444 ? S 15:40 0:00 /usr/sbin/dnsmasq --strict-order --
bind-interfaces --conf-file=
         --domain=novalocal  --pid-file=/var/lib/nova/networks/nova-br100.pid  --
listen-address=192.168.100.1
       --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,static,
120s --dhcp-lease-max=256
               --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf   --dhcp-
script=/usr/bin/nova-dhcpbridge --leasefile-ro
```

If your instances are still not able to obtain IP addresses, the next thing to check is if dnsmasq is seeing the DHCP requests from the instance. On the machine that is running the dnsmasq process, which is the compute host if running in multi-host mode, look at /var/log/syslog to see the dnsmasq output. If dnsmasq is seeing the request properly and handing out an IP, the output looks like:

```
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPDISCOVER(br100) fa:16:3e:56:0b:
6f
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPOFFER(br100) 192.168.100.3 fa:
16:3e:56:0b:6f
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPREQUEST(br100) 192.168.100.3 fa:
16:3e:56:0b:6f
Feb 27 22:01:36 mynode dnsmasq-dhcp[2438]: DHCPACK(br100) 192.168.100.3 fa:
16:3e:56:0b:6f test
```

If you do not see the DHCPDISCOVER, a problem exists with the packet getting from the instance to the machine running dnsmasq. If you see all of above output and your instances are still not able to obtain IP addresses then the packet is able to get from the instance to the host running dnsmasq, but it is not able to make the return trip.

If you see any other message, such as:

```
Feb 27 22:01:36 mynode dnsmasq-dhcp[25435]: DHCPDISCOVER(br100) fa:16:3e:
78:44:84 no address available
```

Then this may be a dnsmasq and/or nova-network related issue. (For the example above, the problem happened to be that dnsmasq did not have any more IP addresses to give away because there were no more Fixed IPs available in the OpenStack Compute database).

If there's a suspicious-looking dnsmasq log message, take a look at the command-line arguments to the dnsmasq processes to see if they look correct.

```
$ ps aux | grep dnsmasq
```

The output looks something like:

```
108 1695 0.0 0.0 25972 1000 ? S Feb26 0:00 /usr/sbin/dnsmasq -u libvirt-dnsmasq
--strict-order --bind-interfaces
    --pid-file=/var/run/libvirt/network/default.pid    --conf-file=    --except-
interface lo --listen-address 192.168.122.1
  --dhcp-range  192.168.122.2,192.168.122.254  --dhcp-leasefile=/var/lib/libvirt/
dnsmasq/default.leases
 --dhcp-lease-max=253 --dhcp-no-override
nobody 2438 0.0 0.0 27540 1096 ? S Feb26 0:00 /usr/sbin/dnsmasq --strict-order
--bind-interfaces --conf-file=
  --domain=novalocal  --pid-file=/var/lib/nova/networks/nova-br100.pid  --listen-
address=192.168.100.1
 --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,static,120s
--dhcp-lease-max=256
 --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/usr/bin/
nova-dhcpbridge --leasefile-ro
root 2439 0.0 0.0 27512 472 ? S Feb26 0:00 /usr/sbin/dnsmasq --strict-order --
bind-interfaces --conf-file=
  --domain=novalocal  --pid-file=/var/lib/nova/networks/nova-br100.pid  --listen-
address=192.168.100.1
 --except-interface=lo --dhcp-range=set:'novanetwork',192.168.100.2,static,120s
--dhcp-lease-max=256
 --dhcp-hostsfile=/var/lib/nova/networks/nova-br100.conf --dhcp-script=/usr/bin/
nova-dhcpbridge --leasefile-ro
```

If the problem does not seem to be related to dnsmasq itself, at this point, use tcpdump on the interfaces to determine where the packets are getting lost.

DHCP traffic uses UDP. The client sends from port 68 to port 67 on the server. Try to boot a new instance and then systematically listen on the NICs until you identify the one that isn't seeing the traffic. To use tcpdump to listen to ports 67 and 68 on br100, you would do:

```
# tcpdump -i br100 -n port 67 or port 68
```

You should be doing sanity checks on the interfaces using command such as "`ip a`" and "`brctl show`" to ensure that the interfaces are actually up and configured the way that you think that they are.

Debugging DNS Issues

If you are able to ssh into an instance, but it takes a very long time (on the order of a minute) to get a prompt, then you might have a DNS issue. The reason a DNS issue can cause this problem is that the ssh server does a reverse DNS lookup on the IP address that you are connecting from. If DNS lookup isn't working on your instances, then you must wait for the DNS reverse lookup timeout to occur for the ssh login process to complete.

When debugging DNS issues, start by making sure the host where the dnsmasq process for that instance runs is able to correctly resolve. If the host cannot resolve, then the instances won't be able either.

A quick way to check if DNS is working is to resolve a hostname inside your instance using the `host` command. If DNS is working, you should see:

```
$ host openstack.org
openstack.org has address 174.143.194.225
openstack.org mail is handled by 10 mx1.emailsrvr.com.
openstack.org mail is handled by 20 mx2.emailsrvr.com.
```

If you're running the Cirros image, it doesn't have the "host" program installed, in which case you can use ping to try to access a machine by hostname to see if it resolves. If DNS is working, the first line of ping would be:

```
$ ping openstack.org
PING openstack.org (174.143.194.225): 56 data bytes
```

If the instance fails to resolve the hostname, you have a DNS problem. For example:

```
$ ping openstack.org
ping: bad address 'openstack.org'
```

In an OpenStack cloud, the dnsmasq process acts as the DNS server for the instances in addition to acting as the DHCP server. A misbehaving dnsmasq process may be the source of DNS-related issues inside the instance. As mentioned in the previous section, the simplest way to rule out a misbehaving dnsmasq process is to kill all of the dnsmasq processes on the machine, and restart nova-network. However, be aware that this command affects everyone running instances on this node, including tenants that have not seen the issue. As a last resort, as root:

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

After the dnsmasq processes start again, check if DNS is working.

If restarting the dnsmasq process doesn't fix the issue, you might need to use tcpdump to look at the packets to trace where the failure is. The DNS server listens on UDP port 53. You should see the DNS request on the bridge (such as, br100) of your compute node. If you start listening with tcpdump on the compute node:

```
# tcpdump -i br100 -n -v udp port 53
tcpdump: listening on br100, link-type EN10MB (Ethernet), capture size 65535
bytes
```

Then, if you ssh into your instance and try to `ping openstack.org`, you should see something like:

```
16:36:18.807518 IP (tos 0x0, ttl 64, id 56057, offset 0, flags [DF], proto UDP
(17), length 59)
 192.168.100.4.54244 > 192.168.100.1.53: 2+ A? openstack.org. (31)
16:36:18.808285 IP (tos 0x0, ttl 64, id 0, offset 0, flags [DF], proto UDP
(17), length 75)
   192.168.100.1.53   >   192.168.100.4.54244:   2   1/0/0   openstack.org.   A
174.143.194.225 (47)
```

CHAPTER 13

Logging and Monitoring

As an OpenStack cloud is composed of so many different services, there are a large number of log files. This section aims to assist you in locating and working with them, and other ways to track the status of your deployment.

Where Are the Logs?

On Ubuntu, most services use the convention of writing their log files to subdirectories of the `/var/log directory`.

Cloud Controller

Service	Log Location
`nova-*`	`/var/log/nova`
`glance-*`	`/var/log/glance`
`cinder-*`	`/var/log/cinder`
`keystone`	`/var/log/keystone`
horizon	`/var/log/apache2/`
misc (Swift, dnsmasq)	`/var/log/syslog`

Compute Nodes

libvirt: `/var/log/libvirt/libvirtd.log`

Console (boot up messages) for VM instances: `/var/lib/nova/instances/instance-<instance id>/console.log`

Block Storage Nodes

cinder: `/var/log/cinder/cinder-volume.log`

How to Read the Logs

OpenStack services use the standard logging levels, at increasing severity: DEBUG, INFO, AUDIT, WARNING, ERROR, CRITICAL, and TRACE. That is, messages only appear in the logs if they are more "severe" than the particular log level with DEBUG allowing all log statements through. For example, TRACE is logged only if the software has a stack trace, while INFO is logged for every message including those that are only for information.

To disable DEBUG-level logging, edit `/etc/nova/nova.conf`:

```
debug=false
```

Keystone is handled a little differently. To modify the logging level, edit the `/etc/keystone/logging.conf` file and look at the `logger_root` and `handler_file` sections.

Logging for Horizon is configured in `/etc/openstack_dashboard/local_settings.py`. As Horizon is a Django web application, it follows the Django Logging (https://docs.djangoproject.com/en/dev/topics/logging/) framework conventions.

The first step in finding the source of an error is typically to search for a CRITICAL, TRACE, or ERROR message in the log starting at the bottom of the log file.

An example of a CRITICAL log message, with the corresponding TRACE (Python traceback) immediately following:

```
2013-02-25 21:05:51 17409 CRITICAL cinder [-] Bad or unexpected response from
the storage volume backend API: volume group
 cinder-volumes doesn't exist
2013-02-25 21:05:51 17409 TRACE cinder Traceback (most recent call last):
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/bin/cinder-volume", line 48,
in <module>
2013-02-25 21:05:51 17409 TRACE cinder service.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
cinder/service.py", line 422, in wait
2013-02-25 21:05:51 17409 TRACE cinder _launcher.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
cinder/service.py", line 127, in wait
2013-02-25 21:05:51 17409 TRACE cinder service.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
```

```
eventlet/greenthread.py", line 166, in wait
2013-02-25 21:05:51 17409 TRACE cinder return self._exit_event.wait()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
eventlet/event.py", line 116, in wait
2013-02-25 21:05:51 17409 TRACE cinder return hubs.get_hub().switch()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
eventlet/hubs/hub.py", line 177, in switch
2013-02-25 21:05:51 17409 TRACE cinder return self.greenlet.switch()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
eventlet/greenthread.py", line 192, in main
2013-02-25 21:05:51 17409 TRACE cinder result = function(*args, **kwargs)
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
cinder/service.py", line 88, in run_server
2013-02-25 21:05:51 17409 TRACE cinder server.start()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
cinder/service.py", line 159, in start
2013-02-25 21:05:51 17409 TRACE cinder self.manager.init_host()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
cinder/volume/manager.py", line 95,
 in init_host
2013-02-25 21:05:51 17409 TRACE cinder self.driver.check_for_setup_error()
2013-02-25 21:05:51 17409 TRACE cinder File "/usr/lib/python2.7/dist-packages/
cinder/volume/driver.py", line 116,
 in check_for_setup_error
2013-02-25 21:05:51 17409 TRACE cinder raise exception.VolumeBackendAPIExcep-
tion(data=exception_message)
2013-02-25 21:05:51 17409 TRACE cinder VolumeBackendAPIException: Bad or unex-
pected response from the storage volume
 backend API: volume group cinder-volumes doesn't exist
2013-02-25 21:05:51 17409 TRACE cinder
```

In this example, cinder-volumes failed to start and has provided a stack trace, since its volume back-end has been unable to setup the storage volume - probably because the LVM volume that is expected from the configuration does not exist.

An example error log:

```
2013-02-25 20:26:33 6619 ERROR nova.openstack.common.rpc.common [-] AMQP server
on localhost:5672 is unreachable:
 [Errno 111] ECONNREFUSED. Trying again in 23 seconds.
```

In this error, a nova service has failed to connect to the RabbitMQ server, because it got a connection refused error.

Tracing Instance Requests

When an instance fails to behave properly, you will often have to trace activity associated with that instance across the log files of various `nova-*` services, and across both the cloud controller and compute nodes.

The typical way is to trace the UUID associated with an instance across the service logs.

Consider the following example:

```
ubuntu@initial:~$ nova list
+--------------------------------+--------+--------+-------------------------+
| ID                             | Name   | Status | Networks                |
+--------------------------------+--------+--------+-------------------------+
| fafed8-4a46-413b-b113-f1959ffe | cirros | ACTIVE | novanetwork=192.168.100.3|
+--------------------------------+--------+--------+-------------------------+
```

Here the ID associated with the instance is `faf7ded8-4a46-413b-b113-f19590746ffe`. If you search for this string on the cloud controller in the `/var/log/nova-*.log` files, it appears in `nova-api.log`, and `nova-scheduler.log`. If you search for this on the compute nodes in `/var/log/nova-*.log`, it appears `nova-network.log` and `nova-compute.log`. If no ERROR or CRITICAL messages appear, the most recent log entry that reports this may provide a hint about what has gone wrong.

Adding Custom Logging Statements

If there is not enough information in the existing logs, you may need to add your own custom logging statements to the `nova-*` services.

The source files are located in `/usr/lib/python2.7/dist-packages/nova`

To add logging statements, the following line should be near the top of the file. For most files, these should already be there:

```
from nova.openstack.common import log as logging
LOG = logging.getLogger(__name__)
```

To add a DEBUG logging statement, you would do:

```
LOG.debug("This is a custom debugging statement")
```

You may notice that all of the existing logging messages are preceded by an underscore and surrounded by parentheses, for example:

```
LOG.debug(_("Logging statement appears here"))
```

This is used to support translation of logging messages into different languages using the gettext (http://docs.python.org/2/library/gettext.html) internationalization library. You don't need to do this for your own custom log messages. However, if you want to contribute the code back to the OpenStack project that includes logging statements, you must surround your log messages with underscore and parentheses.

RabbitMQ Web Management Interface or rabbitmqctl

Aside from connection failures, RabbitMQ log files are generally not useful for debugging OpenStack related issues. Instead, we recommend you use the RabbitMQ web management interface. Enable it on your cloud controller:

```
# /usr/lib/rabbitmq/bin/rabbitmq-plugins enable rabbitmq_management
# service rabbitmq-server restart
```

The RabbitMQ web management interface is accessible on your cloud controller at http://localhost:55672.

Ubuntu 12.04 installs RabbitMQ version 2.7.1, which uses port 55672. RabbitMQ versions 3.0 and above use port 15672 instead. You can check which version of RabbitMQ you have running on your local Ubuntu machine by doing:

```
$ dpkg -s rabbitmq-server | grep "Version:"
Version: 2.7.1-0ubuntu4
```

An alternative to enabling the RabbitMQ Web Management Interface is to use the *rabbitmqctl* commands. For example, *rabbitmqctl list_queues| grep cinder* displays any messages left in the queue. If there are, it's a possible sign that cinder services didn't connect properly to rabbitmq and might have to be restarted.

Items to monitor for RabbitMQ include the number of items in each of the queues and the processing time statistics for the server.

Centrally Managing Logs

Because your cloud is most likely composed of many servers, you must check logs on each of those servers to properly piece an event together. A better solution is to send the logs of all servers to a central location so they can all be accessed from the same area.

Ubuntu uses rsyslog as the default logging service. Since it is natively able to send logs to a remote location, you don't have to install anything extra to enable this feature, just modify the configuration file. In doing this, consider running your logging over a management network, or using an encrypted VPN to avoid interception.

rsyslog Client Configuration

To begin, configure all OpenStack components to log to syslog in addition to their standard log file location. Also configure each component to log to a different syslog facility. This makes it easier to split the logs into individual components on the central server.

`nova.conf`:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL0
```

`glance-api.conf` and `glance-registry.conf`:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL1
```

`cinder.conf`:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL2
```

`keystone.conf`:

```
use_syslog=True
syslog_log_facility=LOG_LOCAL3
```

Swift

By default, Swift logs to syslog.

Next, create /etc/rsyslog.d/client.conf with the following line:

```
*.* @192.168.1.10
```

This instructs rsyslog to send all logs to the IP listed. In this example, the IP points to the Cloud Controller.

rsyslog Server Configuration

Designate a server as the central logging server. The best practice is to choose a server that is solely dedicated to this purpose. Create a file called /etc/rsyslog.d/server.conf with the following contents:

```
# Enable UDP
$ModLoad imudp
# Listen on 192.168.1.10 only
$UDPServerAddress 192.168.1.10
# Port 514
$UDPServerRun 514

# Create logging templates for nova
$template NovaFile,"/var/log/rsyslog/%HOSTNAME%/nova.log"
$template NovaAll,"/var/log/rsyslog/nova.log"



# Log everything else to syslog.log
$template DynFile,"/var/log/rsyslog/%HOSTNAME%/syslog.log"
*.* ?DynFile
```

```
# Log various openstack components to their own individual file
local0.* ?NovaFile
local0.* ?NovaAll
& ~
```

The above example configuration handles the nova service only. It first configures rsyslog to act as a server that runs on port 514. Next, it creates a series of logging templates. Logging templates control where received logs are stored. Using the example above, a nova log from c01.example.com goes to the following locations:

- `/var/log/rsyslog/c01.example.com/nova.log`
- `/var/log/rsyslog/nova.log`

This is useful as logs from c02.example.com go to:

- `/var/log/rsyslog/c02.example.com/nova.log`
- `/var/log/rsyslog/nova.log`

So you have an individual log file for each compute node as well as an aggregated log that contains nova logs from all nodes.

StackTach

StackTach is a tool created by Rackspace to collect and report the notifications sent by `nova`. Notifications are essentially the same as logs, but can be much more detailed. A good overview of notifications can be found at System Usage Data (https://wiki.openstack.org/wiki/SystemUsageData).

To enable nova to send notifications, add the following to `nova.conf`:

```
notification_topics=monitor
notification_driver=nova.openstack.common.notifier.rabbit_notifier
```

Once `nova` is sending notifications, install and configure StackTach. Since StackTach is relatively new and constantly changing, installation instructions would quickly become outdated. Please refer to the StackTach GitHub repo (https://github.com/rackerlabs/stacktach) for instructions as well as a demo video.

Monitoring

There are two types of monitoring: watching for problems and watching usage trends. The former ensures that all services are up and running, creating a functional cloud. The latter involves monitoring resource usage over time in order to make informed decisions about potential bottlenecks and upgrades.

Process Monitoring

A basic type of alert monitoring is to simply check and see if a required process is running. For example, ensure that the `nova-api` service is running on the Cloud Controller:

```
[ root@cloud ~ ] # ps aux | grep nova-api
nova 12786 0.0 0.0 37952 1312 ? Ss Feb11 0:00 su -s /bin/sh -c exec nova-api --
config-file=/etc/nova/nova.conf nova
nova 12787 0.0 0.1 135764 57400 ? S Feb11 0:01 /usr/bin/python /usr/bin/nova-
api --config-file=/etc/nova/nova.conf
nova 12792 0.0 0.0 96052 22856 ? S Feb11 0:01 /usr/bin/python /usr/bin/nova-api
--config-file=/etc/nova/nova.conf
nova 12793 0.0 0.3 290688 115516 ? S Feb11 1:23 /usr/bin/python /usr/bin/nova-
api --config-file=/etc/nova/nova.conf
nova 12794 0.0 0.2 248636 77068 ? S Feb11 0:04 /usr/bin/python /usr/bin/nova-
api --config-file=/etc/nova/nova.conf
root 24121 0.0 0.0 11688 912 pts/5 S+ 13:07 0:00 grep nova-api
```

You can create automated alerts for critical processes by using Nagios and NRPE. For example, to ensure that the `nova-compute` process is running on compute nodes, create an alert on your Nagios server that looks like this:

```
define service {
    host_name c01.example.com
    check_command check_nrpe_1arg!check_nova-compute
    use generic-service
    notification_period 24x7
    contact_groups sysadmins
    service_description nova-compute
}
```

Then on the actual compute node, create the following NRPE configuration:

```
command[check_nova-compute]=/usr/lib/nagios/plugins/check_procs -c 1: -a nova-
compute
```

Nagios checks that at least one nova-compute service is running at all times.

Resource Alerting

Resource alerting provides notifications when one or more resources are critically low. While the monitoring thresholds should be tuned to your specific OpenStack environment, monitoring resource usage is not specific to OpenStack at all – any generic type of alert will work fine.

Some of the resources that you want to monitor include:

- Disk Usage
- Server Load

- Memory Usage
- Network IO
- Available vCPUs

For example, to monitor disk capacity on a compute node with Nagios, add the following to your Nagios configuration:

```
define service {
    host_name c01.example.com
    check_command check_nrpe!check_all_disks!20% 10%
    use generic-service
    contact_groups sysadmins
    service_description Disk
}
```

On the compute node, add the following to your NRPE configuration:

```
command[check_all_disks]=/usr/lib/nagios/plugins/check_disk -w $ARG1$ -c $ARG2$
-e
```

Nagios alerts you with a WARNING when any disk on the compute node is 80% full and CRITICAL when 90% is full.

OpenStack-specific Resources

Resources such as memory, disk, and CPU are generic resources that all servers (even non-OpenStack servers) have and are important to the overall health of the server. When dealing with OpenStack specifically, these resources are important for a second reason: ensuring enough are available in order to launch instances. There are a few ways you can see OpenStack resource usage.

The first is through the `nova` command:

```
# nova usage-list
```

This command displays a list of how many instances a tenant has running and some light usage statistics about the combined instances. This command is useful for a quick overview of your cloud, but doesn't really get into a lot of details.

Next, the `nova` database contains three tables that store usage information.

The `nova.quotas` and `nova.quota_usages` tables store quota information. If a tenant's quota is different than the default quota settings, their quota is stored in `nova.quotas` table. For example:

```
mysql> select project_id, resource, hard_limit from quotas;
+----------------------------------+------------------------------+------------+
| project_id                       | resource                     | hard_limit |
+----------------------------------+------------------------------+------------+
| 628df59f091142399e0689a2696f5baa | metadata_items               | 128        |
| 628df59f091142399e0689a2696f5baa | injected_file_content_bytes  | 10240      |
| 628df59f091142399e0689a2696f5baa | injected_files               | 5          |
| 628df59f091142399e0689a2696f5baa | gigabytes                    | 1000       |
| 628df59f091142399e0689a2696f5baa | ram                          | 51200      |
| 628df59f091142399e0689a2696f5baa | floating_ips                 | 10         |
| 628df59f091142399e0689a2696f5baa | instances                    | 10         |
| 628df59f091142399e0689a2696f5baa | volumes                      | 10         |
| 628df59f091142399e0689a2696f5baa | cores                        | 20         |
+----------------------------------+------------------------------+------------+
```

The `nova.quota_usages` table keeps track of how many resources the tenant currently has in use:

```
mysql> select project_id, resource, in_use from quota_usages where project_id
like '628%';
+----------------------------------+--------------+--------+
| project_id                       | resource     | in_use |
+----------------------------------+--------------+--------+
| 628df59f091142399e0689a2696f5baa | instances    | 1      |
| 628df59f091142399e0689a2696f5baa | ram          | 512    |
| 628df59f091142399e0689a2696f5baa | cores        | 1      |
| 628df59f091142399e0689a2696f5baa | floating_ips | 1      |
| 628df59f091142399e0689a2696f5baa | volumes      | 2      |
| 628df59f091142399e0689a2696f5baa | gigabytes    | 12     |
| 628df59f091142399e0689a2696f5baa | images       | 1      |
+----------------------------------+--------------+--------+
```

By combining the resources used with the tenant's quota, you can figure out a usage percentage. For example, if this tenant is using 1 Floating IP out of 10, then they are using 10% of their Floating IP quota. You can take this procedure and turn it into a formatted report:

```
+----------------------------------+------------+------------+---------------+
| some_tenant                                                               |
+----------------------------------+------------+------------+---------------+
| Resource                         | Used       | Limit      |               |
+----------------------------------+------------+------------+---------------+
| cores                            | 1          | 20         |           5 % |
| floating_ips                     | 1          | 10         |          10 % |
| gigabytes                        | 12         | 1000       |           1 % |
| images                           | 1          | 4          |          25 % |
| injected_file_content_bytes      | 0          | 10240      |           0 % |
| injected_file_path_bytes         | 0          | 255        |           0 % |
| injected_files                   | 0          | 5          |           0 % |
| instances                        | 1          | 10         |          10 % |
| key_pairs                        | 0          | 100        |           0 % |
| metadata_items                   | 0          | 128        |           0 % |
| ram                              | 512        | 51200      |           1 % |
```

```
| reservation_expire                | 0          | 86400      |           0 % |
| security_group_rules              | 0          | 20         |           0 % |
| security_groups                   | 0          | 10         |           0 % |
| volumes                           | 2          | 10         |          20 % |
+-----------------------------------+------------+------------+---------------+
```

The above was generated using a custom script which can be found on GitHub (https://github.com/cybera/novac/blob/dev/libexec/novac-quota-report).

This script is specific to a certain OpenStack installation and must be modified to fit your environment. However, the logic should easily be transferable.

Intelligent Alerting

Intelligent alerting can be thought of as a form of continuous integration for operations. For example, you can easily check to see if Glance is up and running by ensuring that the `glance-api` and `glance-registry` processes are running or by seeing if `glace-api` is responding on port 9292.

But how can you tell if images are being successfully uploaded to the Image Service? Maybe the disk that Image Service is storing the images on is full or the S3 back-end is down. You could naturally check this by doing a quick image upload:

```
#!/bin/bash
#
# assumes that reasonable credentials have been stored at
# /root/auth


. /root/openrc
wget      https://launchpad.net/cirros/trunk/0.3.0/+download/cirros-0.3.0-x86_64-
disk.img
glance   image-create   --name='cirros   image'   --is-public=true   --container-
format=bare --disk-format=qcow2 < cirros-0.3.0-x8
6_64-disk.img
```

By taking this script and rolling it into an alert for your monitoring system (such as Nagios), you now have an automated way of ensuring image uploads to the Image Catalog are working.

You must remove the image after each test. Even better, test whether you can successfully delete an image from the Image Service.

Intelligent alerting takes a considerable more amount of time to plan and implement than the other alerts described in this chapter. A good outline to implement intelligent alerting is:

- Review common actions in your cloud
- Create ways to automatically test these actions
- Roll these tests into an alerting system

Some other examples for Intelligent Alerting include:

- Can instances launch and destroyed?
- Can users be created?
- Can objects be stored and deleted?
- Can volumes be created and destroyed?

Trending

Trending can give you great insight into how your cloud is performing day to day. For example, if a busy day was simply a rare occurrence or if you should start adding new compute nodes.

Trending takes a slightly different approach than alerting. While alerting is interested in a binary result (whether a check succeeds or fails), trending records the current state of something at a certain point in time. Once enough points in time have been recorded, you can see how the value has changed over time.

All of the alert types mentioned earlier can also be used for trend reporting. Some other trend examples include:

- The number of instances on each compute node
- The types of flavors in use
- The number of volumes in use
- The number of Object Storage requests each hour
- The number of nova-api requests each hour
- The I/O statistics of your storage services

As an example, recording `nova-api` usage can allow you to track the need to scale your cloud controller. By keeping an eye on `nova-api` requests, you can determine if you need to spawn more nova-api processes or go as far as introducing an entirely new server to run `nova-api`. To get an approximate count of the requests, look for standard INFO messages in `/var/log/nova/nova-api.log`:

```
# grep INFO /var/log/nova/nova-api.log | wc
```

You can obtain further statistics by looking for the number of successful requests:

```
# grep " 200 " /var/log/nova/nova-api.log | wc
```

By running this command periodically and keeping a record of the result, you can create a trending report over time that shows whether your `nova-api` usage is increasing, decreasing, or keeping steady.

A tool such as collectd can be used to store this information. While collectd is out of the scope of this book, a good starting point would be to use collectd to store the result as a COUNTER data type. More information can be found in collectd's documentation (https://collectd.org/wiki/index.php/Data_source)

CHAPTER 14

Backup and Recovery

Standard backup best practices apply when creating your OpenStack backup policy. For example, how often to backup your data is closely related to how quickly you need to recover from data loss.

If you cannot have any data loss at all, you should focus on High Availability as well as backups.

Other backup considerations include:

- How many backups to keep?
- Should backups be kept off-site?
- How often should backups be tested?

Just as important as a backup policy is a recovery policy (or at least recovery testing).

What to Backup

While OpenStack is composed of many components and moving parts, backing up the critical data is quite simple.

This chapter describes only how to back up configuration files and databases that the various OpenStack components need to run. This chapter does not describe how to back up objects inside Object Storage or data contained inside Block Storage. Generally these areas are left for the user to back up on their own.

Database Backups

The example OpenStack architecture designates the Cloud Controller as the MySQL server. This MySQL server hosts the databases for Nova, Glance, Cinder, and Keystone. With all of these databases in one place, it's very easy to create a database backup:

```
# mysqldump --opt --all-databases >
                openstack.sql
```

If you only want to backup a single database, you can instead run:

```
# mysqldump --opt nova > nova.sql
```

where `nova` is the database you want to back up.

You can easily automate this process by creating a cron job that runs the following script once per day:

```
#!/bin/bash
backup_dir="/var/lib/backups/mysql"
filename="${backup_dir}/mysql-`hostname`-`eval date +%Y%m%d`.sql.gz"
# Dump the entire MySQL database
/usr/bin/mysqldump --opt --all-databases | gzip > $filename
# Delete backups older than 7 days
find $backup_dir -ctime +7 -type f -delete
```

This script dumps the entire MySQL database and delete any backups older than 7 days.

File System Backups

This section discusses which files and directories should be backed up regularly, organized by service.

Compute

The `/etc/nova` directory on both the cloud controller and compute nodes should be regularly backed up.

`/var/log/nova` does not need backed up if you have all logs going to a central area. It is highly recommended to use a central logging server or backup the log directory.

`/var/lib/nova` is another important directory to backup. The exception to this is the `/var/lib/nova/instances` subdirectory on compute nodes. This subdirectory contains the KVM images of running instances. You would only want to back up this directory if you need to maintain backup copies of all instances. Under most circumstances, you do not need to do this, but this can vary from cloud to cloud and your

service levels. Also be aware that making a backup of a live KVM instance can cause that instance to not boot properly if it is ever restored from a backup.

Image Catalog and Delivery

`/etc/glance` and `/var/log/glance` follow the same rules at the nova counterparts.

`/var/lib/glance` should also be backed up. Take special notice of `/var/lib/glance/images`. If you are using a file-based back-end of Glance, `/var/lib/glance/images` is where the images are stored and care should be taken.

There are two ways to ensure stability with this directory. The first is to make sure this directory is run on a RAID array. If a disk fails, the directory is available. The second way is to use a tool such as rsync to replicate the images to another server:

```
# rsync -az --progress /var/lib/glance/images backup-server:/var/lib/glance/images/
```

Identity

`/etc/keystone` and `/var/log/keystone` follow the same rules as other components.

`/var/lib/keystone`, while should not contain any data being used, can also be backed up just in case.

Block Storage

`/etc/cinder` and `/var/log/cinder` follow the same rules as other components.

`/var/lib/cinder` should also be backed up.

Object Storage

`/etc/swift` is very important to have backed up. This directory contains the Swift configuration files as well as the ring files and ring builder files, which if lost render the data on your cluster inaccessible. A best practice is to copy the builder files to all storage nodes along with the ring files. Multiple backups copies are spread throughout your storage cluster.

Recovering Backups

Recovering backups is a fairly simple process. To begin, first ensure that the service you are recovering is not running. For example, to do a full recovery of nova on the cloud controller, first stop all `nova` services:

```
# stop nova-api
      # stop nova-cert
      # stop nova-consoleauth
```

```
      # stop nova-novncproxy
      # stop nova-objectstore
      # stop nova-scheduler
```

Once that's done, stop MySQL:

```
# stop mysql
```

Now you can import a previously backed up database:

```
# mysql nova < nova.sql
```

As well as restore backed up nova directories:

```
# mv /etc/nova{,.orig}
      # cp -a /path/to/backup/nova /etc/
```

Once the files are restored, start everything back up:

```
# start mysql
        # for i in nova-api nova-cert nova-consoleauth nova-novncproxy nova-
objectstore nova-scheduler
      > do
      > start $i
      > done
```

Other services follow the same process, with their respective directories and databases.

CHAPTER 15

Customize

OpenStack might not do everything you need it to do out of the box. In these cases, you can follow one of two major paths. First, you can learn How To Contribute (https://wiki.openstack.org/wiki/How_To_Contribute), follow the Code Review Workflow (https://wiki.openstack.org/wiki/GerritWorkflow), make your changes and contribute them back to the upstream OpenStack project. This path is recommended if the feature you need requires deep integration with an existing project. The community is always open to contributions and welcomes new functionality that follows the feature development guidelines.

Alternately, if the feature you need does not require deep integration, there are other ways to customize OpenStack. If the project where your feature would need to reside uses the Python Paste framework, you can create middleware for it and plug it in through configuration. There may also be specific ways of customizing an project such as creating a new scheduler for OpenStack Compute or a customized Dashboard. This chapter focuses on the second method of customizing OpenStack.

To customize OpenStack this way you'll need a development environment. The best way to get an environment up and running quickly is to run DevStack within your cloud.

DevStack

You can find all of the documentation at the DevStack (http://devstack.org/) website. Depending on which project you would like to customize, either Object Storage (swift) or another project, you must configure DevStack differently. For the middleware example below, you must install with the Object Store enabled.

To run DevStack for the stable Folsom branch on an instance:

1. Boot an instance from the Dashboard or the nova command-line interface (CLI) with the following parameters.

   - Name: devstack
   - Image: Ubuntu 12.04 LTS
   - Memory Size: 4 GB RAM (you could probably get away with 2 GB)
   - Disk Size: minimum 5 GB

   If you are using the `nova` client, specify `--flavor 6` on the `nova boot` command to get adequate memory and disk sizes.

2. If your images have only a root user, you must create a "stack" user. Otherwise you run into permission issues with screen if you let `stack.sh` create the "stack" user for you. If your images already have a user other than root, you can skip this step.

   a. `ssh root@<IP Address>`

   b. `adduser --gecos "" stack`

   c. Enter a new password at the prompt.

   d. `adduser stack sudo`

   e. `grep -q "^#includedir.*/etc/sudoers.d" /etc/sudoers || echo "#in cludedir /etc/sudoers.d" >> /etc/sudoers`

   f. `( umask 226 && echo "stack ALL=(ALL) NOPASSWD:ALL" > /etc/sudo ers.d/50_stack_sh )`

   g. `exit`

3. Now login as the stack user and set up DevStack.

   a. `ssh stack@<IP address>`

   b. At the prompt, enter the password that you created for the stack user.

   c. `sudo apt-get -y update`

   d. `sudo apt-get -y install git`

   e. `git clone https://github.com/openstack-dev/devstack.git -b sta ble/folsom devstack/`

   f. `cd devstack`

   g. `vim localrc`

      a. For Swift only, used in the **Middleware Example**, see the example **[1] Swift only localrc** below

b. For all other projects, used in the **Nova Scheduler Example**, see the example **[2] All other projects localrc** below

h. `./stack.sh`

i. `screen -r stack`

- The `stack.sh` script takes a while to run. Perhaps take this opportunity to join the OpenStack foundation (http://www.openstack.org/join/).
- When you run `stack.sh`, you might see an error message that reads "ERROR: at least one RPC back-end must be enabled". Don't worry about it; swift and keystone do not need an RPC (AMQP) back-end. You can also ignore any `ImportErrors`.
- Screen is a useful program for viewing many related services at once. For more information, see GNU screen quick reference. (http://aperiodic.net/screen/quick_reference)

Now that you have an OpenStack development environment, you're free to hack around without worrying about damaging your production deployment. Proceed to either the **Middleware Example** for a Swift-only environment, or the **Nova Scheduler Example** for all other projects.

[1] Swift only localrc

```
ADMIN_PASSWORD=devstack
MYSQL_PASSWORD=devstack
RABBIT_PASSWORD=devstack
SERVICE_PASSWORD=devstack
SERVICE_TOKEN=devstack

SWIFT_HASH=66a3d6b56c1f479c8b4e70ab5c2000f5
SWIFT_REPLICAS=1

# Uncomment the BRANCHes below to use stable versions


# unified auth system (manages accounts/tokens)
KEYSTONE_BRANCH=stable/folsom
# object storage
SWIFT_BRANCH=stable/folsom

disable_all_services
enable_service key swift mysql
```

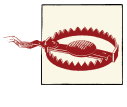
[2] All other projects localrc

```
ADMIN_PASSWORD=devstack
MYSQL_PASSWORD=devstack
RABBIT_PASSWORD=devstack
SERVICE_PASSWORD=devstack
SERVICE_TOKEN=devstack

FLAT_INTERFACE=br100
PUBLIC_INTERFACE=eth0

VOLUME_BACKING_FILE_SIZE=20480M

# For stable versions, look for branches named stable/[milestone].

# compute service
NOVA_BRANCH=stable/folsom

# volume service
CINDER_BRANCH=stable/folsom

# image catalog service
GLANCE_BRANCH=stable/folsom

# unified auth system (manages accounts/tokens)
KEYSTONE_BRANCH=stable/folsom

# django powered web control panel for openstack
HORIZON_BRANCH=stable/folsom
```

Middleware Example

Most OpenStack projects are based on the Python Paste(http://pythonpaste.org/) framework. The best introduction to its architecture is A Do-It-Yourself Framework (http://pythonpaste.org/do-it-yourself-framework.html). Due to the use of this framework, you are able to add features to a project by placing some custom code in a project's pipeline without having to change any of the core code.

To demonstrate customizing OpenStack like this, we'll create a piece of middleware for swift that allows access to a container from only a set of IP addresses, as determined by the container's metadata items. Such an example could be useful in many contexts. For example, you might have public access to one of your containers, but what you really want to restrict it to is a set of IPs based on a whitelist.



This example is for illustrative purposes only. It should not be used as a container IP whitelist solution without further development and extensive security testing.

When you join the screen session that `stack.sh` starts with `screen -r stack`, you're greeted with three screens if you used the localrc file with just Swift installed.

```
0$ shell*  1$ key  2$ swift
```

The asterisk * indicates which screen you are on.

- **`0$ shell`** . A shell where you can get some work done.
- **`1$ key`** . The keystone service.
- **`2$ swift`** . The swift proxy service.

To create the middleware and plug it in through Paste configuration:

1. All of the code for OpenStack lives in `/opt/stack`. Go to the swift directory in the shell screen and edit your middleware module.
   a. `cd /opt/stack/swift`
   b. `vim swift/common/middleware/ip_whitelist.py`
2. Copy in the following code. When you're done, save and close the file.

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

#    http://www.apache.org/licenses/LICENSE-2.0
#
# Unless required by applicable law or agreed to in writing, software
# distributed under the License is distributed on an "AS IS" BASIS,
# WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or
# implied.
# See the License for the specific language governing permissions and
# limitations under the License.
import socket

from swift.common.utils import get_logger
from swift.proxy.controllers.base import get_container_info
from swift.common.swob import Request, Response

class IPWhitelistMiddleware(object):
    """
    IP Whitelist Middleware

    Middleware that allows access to a container from only a set of IP
    addresses as determined by the container's metadata items that start
    with the prefix 'allow'. E.G. allow-dev=192.168.0.20
    """

    def __init__(self, app, conf, logger=None):
        self.app = app
```

```
        if logger:
            self.logger = logger
        else:
            self.logger = get_logger(conf, log_route='ip_whitelist')

        self.deny_message = conf.get('deny_message', "IP Denied")
        self.local_ip = socket.gethostbyname(socket.gethostname())

    def __call__(self, env, start_response):
        """
        WSGI entry point.
        Wraps env in swob.Request object and passes it down.

        :param env: WSGI environment dictionary
        :param start_response: WSGI callable
        """
        req = Request(env)

        try:
            version, account, container, obj = req.split_path(1, 4, True)
        except ValueError:
            return self.app(env, start_response)

        container_info = get_container_info(
            req.environ, self.app, swift_source='IPWhitelistMiddleware')

        remote_ip = env['REMOTE_ADDR']
        self.logger.debug(_("Remote IP: %(remote_ip)s"),
                          {'remote_ip': remote_ip})

        meta = container_info['meta']
        allow = {k:v for k,v in meta.iteritems() if k.startswith('allow')}
        allow_ips = set(allow.values())
        allow_ips.add(self.local_ip)
        self.logger.debug(_("Allow IPs: %(allow_ips)s"),
                          {'allow_ips': allow_ips})

        if remote_ip in allow_ips:
            return self.app(env, start_response)
        else:
            self.logger.debug(
                _("IP %(remote_ip)s denied access to Account=%(account)s "
                       "Container=%(container)s. Not in %(allow_ips)s"), lo
cals())
            return Response(
                status=403,
                body=self.deny_message,
                request=req)(env, start_response)


def filter_factory(global_conf, **local_conf):
```

```
    """
    paste.deploy app factory for creating WSGI proxy apps.
    """
    conf = global_conf.copy()
    conf.update(local_conf)

    def ip_whitelist(app):
        return IPWhitelistMiddleware(app, conf)
    return ip_whitelist
```

There is a lot of useful information in `env` and `conf` that you can use to decide what to do with the request. To find out more about what properties are available, you can insert the following log statement into the `__init__` method

```
self.logger.debug(_("conf = %(conf)s"), locals())
```

and the following log statement into the `__call__` method

```
self.logger.debug(_("env = %(env)s"), locals())
```

3. To plug this middleware into the Swift pipeline you'll need to edit one configuration file.

```
vim /etc/swift/proxy-server.conf
```

4. Find the `[filter:ratelimit]` section and copy in the following configuration section.

```
[filter:ip_whitelist]
paste.filter_factory = swift.common.middleware.ip_whitelist:filter_factory
# You can override the default log routing for this filter here:
# set log_name = ratelimit
# set log_facility = LOG_LOCAL0
# set log_level = INFO
# set log_headers = False
# set log_address = /dev/log
deny_message = You shall not pass!
```

5. Find the `[pipeline:main]` section and add `ip_whitelist` to the list like so. When you're done, save and close the file.

```
[pipeline:main]
pipeline = catch_errors healthcheck cache ratelimit ip_whitelist authtoken
keystoneauth proxy-logging proxy-server
```

6. Restart the Swift Proxy service to make Swift use your middleware. Start by switching to the swift screen.
   a. Press Ctrl-A followed by pressing 2, where 2 is the label of the screen. You can also press Ctrl-A followed by pressing n to go to the next screen.
   b. Press Ctrl-C to kill the service.
   c. Press Up Arrow to bring up the last command.

d. Press Enter to run it.

7. Test your middleware with the Swift CLI. Start by switching to the shell screen and finish by switching back to the swift screen to check the log output.
   a. Press Ctrl-A followed by pressing 0
   b. `cd ~/devstack`
   c. `source openrc`
   d. `swift post middleware-test`
   e. Press Ctrl-A followed by pressing 2
8. Among the log statements you'll see the lines.

```
proxy-server ... IPWhitelistMiddleware
proxy-server Remote IP: 203.0.113.68 (txn: ...)
proxy-server Allow IPs: set(['203.0.113.68']) (txn: ...)
```

   The first three statements basically have to do with the fact that middleware doesn't need to re-authenticate when it interacts with other Swift services. The last 2 statements are produced by our middleware and show that the request was sent from our DevStack instance and was allowed.
9. Test the middleware from outside of DevStack on a remote machine that has access to your DevStack instance.
   a. `swift --os-auth-url=http://203.0.113.68:5000/v2.0/ --os-region-name=RegionOne --os-username=demo:demo --os-password=devstack list middleware-test`
   b. Container GET failed: http://203.0.113.68:8080/v1/AUTH_.../middleware-test?format=json 403 Forbidden You shall not pass!
10. Check the Swift log statements again and among the log statements you'll see the lines.

```
proxy-server Invalid user token - deferring reject downstream
proxy-server Authorizing from an overriding middleware (i.e: tempurl)
(txn: ...)
proxy-server ... IPWhitelistMiddleware
proxy-server Remote IP: 198.51.100.12 (txn: ...)
proxy-server Allow IPs: set(['203.0.113.68']) (txn: ...)
proxy-server IP 198.51.100.12 denied access to Account=AUTH_... Contain-
er=None. Not in set(['203.0.113.68']) (txn: ...)
```

    Here we can see that the request was denied because the remote IP address wasn't in the set of allowed IPs.
11. Back on your DevStack instance add some metadata to your container to allow the request from the remote machine.
    a. Press Ctrl-A followed by pressing 0

b. `swift post --meta allow-dev:198.51.100.12 middleware-test`

12. Now try the command from ??? again and it succeeds.

Functional testing like this is not a replacement for proper unit and integration testing but it serves to get you started.

A similar pattern can be followed in all other projects that use the Python Paste framework. Simply create a middleware module and plug it in through configuration. The middleware runs in sequence as part of that project's pipeline and can call out to other services as necessary. No project core code is touched. Look for a `pipeline` value in the project's `conf` or `ini` configuration files in `/etc/<project>` to identify projects that use Paste.

When your middleware is done, we encourage you to open source it and let the community know on the OpenStack mailing list. Perhaps others need the same functionality. They can use your code, provide feedback, and possibly contribute. If enough support exists for it, perhaps you can propose that it be added to the official Swift middleware (https://github.com/openstack/swift/tree/master/swift/common/middleware).

Nova Scheduler Example

Many OpenStack projects allow for customization of specific features using a driver architecture. You can write a driver that conforms to a particular interface and plug it in through configuration. For example, you can easily plug in a new scheduler for nova. The existing schedulers for nova are feature full and well documented at Scheduling (http://docs.openstack.org/trunk/config-reference/content/section_compute-scheduler.html). However, depending on your user's use cases, the existing schedulers might not meet your requirements. You might need to create a new scheduler.

To create a scheduler you must inherit from the class `nova.scheduler.driver.Scheduler`. Of the five methods that you can override, you *must* override the two methods indicated with a "*" below.

- `update_service_capabilities`
- `hosts_up`
- `schedule_live_migration`
- * `schedule_prep_resize`
- * `schedule_run_instance`

To demonstrate customizing OpenStack, we'll create an example of a nova scheduler that randomly places an instance on a subset of hosts depending on the originating IP

address of the request and the prefix of the hostname. Such an example could be useful when you have a group of users on a subnet and you want all of their instances to start within some subset of your hosts.

This example is for illustrative purposes only. It should not be used as a scheduler for Nova without further development and testing.

When you join the screen session that `stack.sh` starts with `screen -r stack`, you are greeted with many screens.

```
0$ shell*  1$ key  2$ g-reg  3$ g-api  4$ n-api  5$ n-cpu  6$ n-crt  7$ n-net
 8-$ n-sch ...
```

- **`shell`** . A shell where you can get some work done.
- **`key`** . The keystone service.
- **`g-*`** . The glance services.
- **`n-*`** . The nova services.
- **`n-sch`** . The nova scheduler service.

To create the scheduler and plug it in through configuration:

1. The code for OpenStack lives in `/opt/stack` so go to the nova directory and edit your scheduler module.
   a. `cd /opt/stack/nova`
   b. `vim nova/scheduler/ip_scheduler.py`
2. Copy in the following code. When you're done, save and close the file.

```
# vim: tabstop=4 shiftwidth=4 softtabstop=4
# Copyright (c) 2013 OpenStack Foundation
# All Rights Reserved.
#
#    Licensed under the Apache License, Version 2.0 (the "License"); you may
#    not use this file except in compliance with the License. You may obtain
#    a copy of the License at
#
#   http://www.apache.org/licenses/LICENSE-2.0
#
#    Unless required by applicable law or agreed to in writing, software
#      distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT
#     WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See
the
#    License for the specific language governing permissions and limitations
#    under the License.
```

```
"""
IP Scheduler implementation
"""

import random

from nova import exception
from nova.openstack.common import log as logging
from nova import flags
from nova.scheduler import driver

FLAGS = flags.FLAGS
LOG = logging.getLogger(__name__)


class IPScheduler(driver.Scheduler):
    """
    Implements Scheduler as a random node selector based on
    IP address and hostname prefix.
    """

    def _filter_hosts(self, hosts, hostname_prefix):
        """Filter a list of hosts based on hostname prefix."""

        hosts = [host for host in hosts if host.startswith(hostname_prefix)]
        return hosts

    def _schedule(self, context, topic, request_spec, filter_properties):
        """
        Picks a host that is up at random based on
        IP address and hostname prefix.
        """

        elevated = context.elevated()
        hosts = self.hosts_up(elevated, topic)

        if not hosts:
            msg = _("Is the appropriate service running?")
            raise exception.NoValidHost(reason=msg)

        remote_ip = context.remote_address

        if remote_ip.startswith('10.1'):
            hostname_prefix = 'doc'
        elif remote_ip.startswith('10.2'):
            hostname_prefix = 'ops'
        else:
            hostname_prefix = 'dev'

        hosts = self._filter_hosts(hosts, hostname_prefix)
        host = hosts[int(random.random() * len(hosts))]
```

```
            LOG.debug(_("Request from %(remote_ip)s scheduled to %(host)s")
                      % locals())

            return host

        def schedule_run_instance(self, context, request_spec,
                                  admin_password, injected_files,
                                  requested_networks, is_first_time,
                                  filter_properties):
            """Attempts to run the instance"""
            instance_uuids = request_spec.get('instance_uuids')
            for num, instance_uuid in enumerate(instance_uuids):
                request_spec['instance_properties']['launch_index'] = num
                try:
                    host = self._schedule(context, 'compute', request_spec,
                                          filter_properties)
                    updated_instance = driver.instance_update_db(context,
                                                                 instance_uuid)
                    self.compute_rpcapi.run_instance(context,
                                                     instance=updated_instance,
host=host,
                                                                 requested_net
works=requested_networks,
                                                          injected_files=injec
ted_files,
                                                       admin_password=admin_pass
word,

is_first_time=is_first_time,
                                                    request_spec=request_spec,
                                                                 filter_proper
ties=filter_properties)
                except Exception as ex:
                     # NOTE(vish): we don't reraise the exception here to make
sure
                    # that all instances in the request get set to
                    # error properly
                    driver.handle_schedule_error(context, ex, instance_uuid,
                                                 request_spec)

        def schedule_prep_resize(self, context, image, request_spec,
                                 filter_properties, instance, instance_type,
                                 reservations):
            """Select a target for resize."""
            host = self._schedule(context, 'compute', request_spec,
                                  filter_properties)
            self.compute_rpcapi.prep_resize(context, image, instance,
                                            instance_type, host, reservations)
```

There is a lot of useful information in `context`, `request_spec`, and `filter_properties` that you can use to decide where to schedule the instance. To find out

more about what properties are available you can insert the following log statements into the `schedule_run_instance` method of the scheduler above.

```
LOG.debug(_("context = %(context)s") % {'context': context.__dict__})LOG.de-
bug(_("request_spec = %(request_spec)s") % locals())LOG.debug(_("fil-
ter_properties = %(filter_properties)s") % locals())
```

3. To plug this scheduler into Nova you'll need to edit one configuration file.

```
LOG$ vim /etc/nova/nova.conf
```

4. Find the `compute_scheduler_driver` config and change it like so.

```
LOGcompute_scheduler_driver=nova.scheduler.ip_scheduler.IPScheduler
```

5. Restart the Nova scheduler service to make Nova use your scheduler. Start by switching to the `n-sch` screen.
   a. Press Ctrl-A followed by pressing 8
   b. Press Ctrl-C to kill the service
   c. Press Up Arrow to bring up the last command
   d. Press Enter to run it
6. Test your scheduler with the Nova CLI. Start by switching to the shell screen and finish by switching back to the `n-sch` screen to check the log output.
   a. Press Ctrl-A followed by pressing 0
   b. `cd ~/devstack`
   c. `source openrc`
   d. `IMAGE_ID=`nova image-list | egrep cirros | egrep -v "kernel|ram disk" | awk '{print $2}'``
   e. `nova boot --flavor 1 --image $IMAGE_ID scheduler-test`
   f. Press Ctrl-A followed by pressing 8
7. Among the log statements you'll see the line.

```
LOG2013-02-27 17:39:31 DEBUG nova.scheduler.ip_scheduler [req-... demo
demo] Request from 50.56.172.78 scheduled to
devstack-nova from (pid=4118) _schedule /opt/stack/nova/nova/scheduler/
ip_scheduler.py:73
```

Functional testing like this is not a replacement for proper unit and integration testing but it serves to get you started.

A similar pattern can be followed in all other projects that use the driver architecture. Simply create a module and class that conform to the driver interface and plug it in through configuration. Your code runs when that feature is used and can call out to other services as necessary. No project core code is touched. Look for a "driver" value

in the project's conf configuration files in `/etc/<project>` to identify projects that use a driver architecture.

When your scheduler is done, we encourage you to open source it and let the community know on the OpenStack mailing list. Perhaps others need the same functionality. They can use your code, provide feedback, and possibly contribute. If enough support exists for it, perhaps you can propose that it be added to the official Nova schedulers (https://github.com/openstack/nova/tree/master/nova/scheduler).

Dashboard

The Dashboard is based on the Python Django (https://www.djangoproject.com/) web application framework. The best guide to customizing it has already been written and can be found at Build on Horizon (http://docs.openstack.org/developer/horizon/topics/tutorial.html).

CHAPTER 16

Upstream OpenStack

OpenStack is founded on a thriving community which is a source of help, and welcomes your contributions. This section details some of the ways you can interact with the others involved.

Getting Help

There are several avenues available for seeking assistance. The quickest way to is to help the community help you. Search the Q&A sites, mailing list archives, and bug lists for issues similar to yours. If you can't find anything, follow the directions for Reporting Bugs in the section below or use one of the channels for support below.

Your first port of call should be the official OpenStack documentation, found on http://docs.openstack.org.

You can get questions answered on the ask.openstack.org site.

Mailing Lists (https://wiki.openstack.org/wiki/Mailing_Lists) are also a great place to get help. The wiki page has more information about the various lists. As an operator, the main lists you should be aware of are:

- General list: `openstack@lists.openstack.org`. The scope of this list is the current state of OpenStack. This is a very high traffic mailing list, with many, many emails per day.
- Operators list: `openstack-operators@lists.openstack.org`. This list is intended for discussion among existing OpenStack cloud operators, such as yourself. Currently, this list is relatively low traffic, on the order of one email a day.

- Development list: `openstack-dev@lists.openstack.org`. The scope of this list is the future state of OpenStack. This is a high traffic mailing list, with multiple emails per day.

We recommend you subscribe to the general list and the operator list, although you must set up filters to manage the volume for the general list. You'll also find links to the mailing list archives on the mailing list wiki page where you can search through the discussions.

Multiple IRC channels (https://wiki.openstack.org/wiki/IRC) are available for general questions and developer discussions. The general discussion channel is `#openstack` on `irc.freenode.net`.

Reporting Bugs

As an operator, you are in a very good position to report unexpected behavior with your cloud. As OpenStack is flexible, you may be the only individual to report a particular issue. Every issue is important to fix so it is essential to learn how to easily submit a bug report.

All OpenStack projects use Launchpad for bug tracking. You'll need to create an account on Launchpad before you can submit a bug report.

Once you have a Launchpad account, reporting a bug is as simple as identifying the project, or projects that are causing the issue. Sometimes this is more difficult than expected, but those working on the bug triage are happy to help relocate issues if their not in the right place initially.

- Report a bug in Nova (https://bugs.launchpad.net/nova/+filebug)
- Report a bug in python-novaclient (https://bugs.launchpad.net/python-novaclient/+filebug)
- Report a bug in Swift (https://bugs.launchpad.net/swift/+filebug)
- Report a bug in python-swiftclient (https://bugs.launchpad.net/python-swiftclient/+filebug)
- Report a bug in Glance (https://bugs.launchpad.net/glance/+filebug)
- Report a bug in python-glanceclient (https://bugs.launchpad.net/python-glanceclient/+filebug)
- Report a bug in Keystone (https://bugs.launchpad.net/keystone/+filebug)
- Report a bug in python-keystoneclient (https://bugs.launchpad.net/python-keystoneclient/+filebug)
- Report a bug in Quantum (https://bugs.launchpad.net/quantum/+filebug)

- Report a bug in python-quantumclient (https://bugs.launchpad.net/python-quantumclient/+filebug)
- Report a bug in Cinder (https://bugs.launchpad.net/cinder/+filebug)
- Report a bug in python-cinderclient (https://bugs.launchpad.net/python-cinderclient/+filebug)
- Report a bug in Horizon (https://bugs.launchpad.net/horizon/+filebug)
- Report a bug with the documentation (http://bugs.launchpad.net/openstack-manuals/+filebug)
- Report a bug with the API documentation (http://bugs.launchpad.net/openstack-api-site/+filebug)

To write a good bug report, the following process is essential. First, search for the bug to make sure there is no bug already filed for the same issue. If you find one, be sure to click on "This bug affects X people. Does this bug affect you?" If you can't find the issue then enter the details of your report. It should at least include:

- The release, or milestone, or commit ID corresponding to the software that you are running.
- The operating system and version where you've identified the bug.
- Steps to reproduce the bug, including what went wrong.
- Description of the expected results instead of what you saw.
- Read and understood your log files so you only include relevant excerpts.

When you do this, the bug is created with:

- Status: *New*

In the bug comments, you can contribute instructions on how to fix a given bug, and set it to *Triaged*. Or you can directly fix it: assign the bug to yourself, set it to *In progress*, branch the code, implement the fix, and propose your change for merging into trunk. But let's not get ahead of ourselves, there are bug triaging tasks as well.

Confirming & Prioritizing

This stage is about checking that a bug is real and assessing its impact. Some of these steps require bug supervisor rights (usually limited to core teams). If the bug lacks information to properly reproduce or assess the importance of the bug, the bug is set to:

- Status: *Incomplete*

Once you have reproduced the issue (or are 100% confident that this is indeed a valid bug) and have permissions to do so, set:

- Status: *Confirmed*

Core developers also prioritize the bug, based on its impact:

- Importance: <Bug impact>

The bug impacts are categorized as follows:

1. *Critical* if the bug prevents a key feature from working properly (regression) for all users (or without a simple workaround) or result in data loss
2. *High* if the bug prevents a key feature from working properly for some users (or with a workaround)
3. *Medium* if the bug prevents a secondary feature from working properly
4. *Low* if the bug is mostly cosmetic
5. *Wishlist* if the bug is not really a bug, but rather a welcome change in behavior

If the bug contains the solution, or a patch, set the bug status to *Triaged*

Bug Fixing

At this stage, a developer works on a fix. During that time, to avoid duplicating the work, they should set:

- Status: *In progress*
- Assignee: <yourself>

When the fix is ready, they propose and get the change reviewed.

After the Change is Accepted

After the change is reviewed, accepted, and lands in master, it automatically moves to:

- Status: *Fix committed*

When the fix makes it into a milestone or release branch, it automatically moves to:

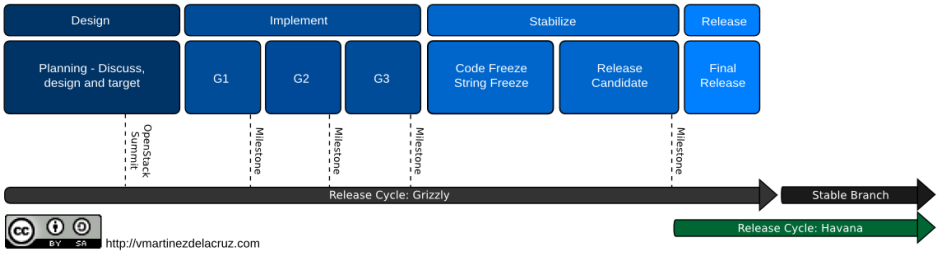
- Milestone: Milestone the bug was fixed in
- Status: *Fix released*

Join the OpenStack Community

Since you've made it this far in the book, you should consider becoming an official individual member of the community and Join The OpenStack Foundation (https://www.openstack.org/join/). The OpenStack Foundation is an independent body providing shared resources to help achieve the OpenStack mission by protecting, empowering, and promoting OpenStack software and the community around it, including users, developers and the entire ecosystem. We all share the responsibility to make this community the best it can possibly be and signing up to be a member is the first step to participating. Like the software, individual membership within the OpenStack Foundation is free and accessible to anyone.

Features and the Development Roadmap

OpenStack follows a six month release cycle, typically releasing in April and October each year. At the start of each cycle, the community gathers in a single location for a Design Summit. At the summit, the features for the coming releases are discussed, prioritized and planned. Here's an example release cycle with dates showing milestone releases, code freeze, and string freeze dates along with an example of when the Summit occurs. Milestones are interim releases within the cycle that are available as packages for download and testing. Code freeze is putting a stop to adding new features to the release. String freeze is putting a stop to changing any strings within the source code.



Feature requests typically start their life in Etherpad, a collaborative editing tool, which is used to take coordinating notes at a design summit session specific to the feature. This then leads to the creation of a blueprint on the Launchpad site for the particular project, which is used to describe the feature more formally. Blueprints are then approved by project team members, and development can begin.

Therefore, the fastest way to get your feature request up for consideration is to create an Etherpad with your ideas and propose a session to the design summit. If the design summit has already passed, you may also create a blueprint directly. Read this blog post about how to work with blueprints (http://vmartinezdelacruz.com/how-to-

work-with-blueprints-without-losing-your-mind/) for a developer intern's perspective, Victoria Martínez.

The roadmap for the next release as it is developed can be seen at Releases (http://status.openstack.org/release/).

To determine the potential features going in to future releases, or to look at features implemented previously, take a look at the existing blueprints such as OpenStack Compute (nova) Blueprints (https://blueprints.launchpad.net/nova), OpenStack Identity (keystone) Blueprints (https://blueprints.launchpad.net/keystone) and release notes.

Release notes are maintained on the OpenStack wiki:

Series	Status	Releases	Date
Grizzly	Under development, Release schedule	Due	Apr 4, 2013
Folsom	Current stable release, security-supported	2012.2	Sep 27, 2012
2012.2.1	Nov 29, 2012		
2012.2.2	Dec 13, 2012		
2012.2.3	Jan 31, 2012		
Essex	Community-supported, security-supported	2012.1	Apr 5, 2012
2012.1.1	Jun 22, 2012		
2012.1.2	Aug 10, 2012		
2012.1.3	Oct 12, 2012		
Diablo	Community-supported	2011.3	Sep 22, 2011
2011.3.1	Jan 19, 2012		
Cactus	Deprecated	2011.2	Apr 15, 2011
Bexar	Deprecated	2011.1	Feb 3, 2011
Austin	Deprecated	2010.1	Oct 21, 2010

How to Contribute to the Documentation

OpenStack documentation efforts encompass operator and administrator docs, API docs, and user docs.

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

Just like the code, the docs.openstack.org site is updated constantly using the Gerrit review system, with source stored in GitHub in the openstack-manuals (http://github.com/openstack/openstack-manuals/) repository and the api-site (http://github.com/openstack/api-site/) repository, in DocBook format.

To review the documentation before it's published, go to the OpenStack Gerrit server at review.openstack.org and search for project:openstack/openstack-manuals or project:openstack/api-site.

See the How To Contribute (https://wiki.openstack.org/wiki/How_To_Contribute) page on the wiki for more information on the steps you need to take to submit your first documentation review or change.

Security Information

As a community, we take security very seriously and follow a specific process for reporting potential issues. We vigilantly pursue fixes and regularly eliminate exposures. You can report security issues you discover through this specific process. The OpenStack Vulnerability Management Team is a very small group of experts in vulnerability management drawn from the OpenStack community. Their job is facilitating the reporting of vulnerabilities, coordinating security fixes and handling progressive disclosure of the vulnerability information. Specifically, the Team is responsible for the following functions:

- Vulnerability Management: All vulnerabilities discovered by community members (or users) can be reported to the Team.
- Vulnerability Tracking: The Team will curate a set of vulnerability related issues in the issue tracker. Some of these issues are private to the Team and the affected product leads, but once remediation is in place, all vulnerabilities are public.
- Responsible Disclosure: As part of our commitment to work with the security community, the team ensures that proper credit is given to security researchers who responsibly report issues in OpenStack.

We provide two ways to report issues to the OpenStack Vulnerability Management Team depending on how sensitive the issue is:

- Open a bug in Launchpad and mark it as a 'security bug'. This makes the bug private and accessible to only the Vulnerability Management Team.
- If the issue is extremely sensitive, send an encrypted email to one of the Team's members. Find their GPG keys at OpenStack Security (http://www.openstack.org/projects/openstack-security/).

You can find the full list of security-oriented teams you can join at Security Teams (http://wiki.openstack.org/SecurityTeams). The Vulnerability Management process is fully documented at Vulnerability Management (https://wiki.openstack.org/wiki/VulnerabilityManagement).

Finding Additional Information

In addition to this book, there are many other sources of information about OpenStack. The OpenStack website (http://www.openstack.org) is a good starting point, with OpenStack Docs (http://docs.openstack.org) and OpenStack API Docs (http://api.openstack.org) providing technical documentation about OpenStack. The OpenStack wiki contains a lot of general information that cuts across the OpenStack projects including a list of recommended tools (https://wiki.openstack.org/wiki/OperationsTools). Finally, there are a number of blogs aggregated at Planet OpenStack (http://planet.openstack.org).

CHAPTER 17

Advanced Configuration

OpenStack is intended to work well across a variety of installation flavors, from very small private clouds to large public clouds. In order to achieve this the developers add configuration options to their code which allow the behaviour of the various components to be tweaked depending on your needs. Unfortunately it is not possible to cover all possible deployments with the default configuration values.

At the time of writing, OpenStack has over 1,500 configuration options. You can see them documented at the OpenStack configuration reference guide. This chapter cannot hope to document all of these, but however we do try to introduce the important concepts so that you know where to go digging for more information.

Differences between various drivers

Many OpenStack projects implement a driver layer, and each of these drivers will implement their own configuration options. For example in OpenStack Compute (Nova), there are various hypervisor drivers implemented -- libvirt, xenserver, hyper-v and vmware for example. Not all of these hypervisor drivers have the same features, and each has different tuning requirements.

The currently implemented hypervisors are listed on the OpenStack documentation website. You can see a matrix of the various features in OpenStack Compute (Nova) hypervisor drivers on the OpenStack wiki at the Hypervisor support matrix page.

The point we are trying to make here is that just because an option exists doesn't mean that option is relevant to your driver choices. Normally the documentation notes which drivers the configuration applies to.

Periodic tasks

Another common concept across various OpenStack projects is that of periodic tasks. Periodic tasks are much like cron jobs on traditional Unix systems, but they are run inside of an OpenStack process. For example, when OpenStack Compute (Nova) needs to work out what images it can remove from its local cache, it runs a periodic task to do this.

Periodic tasks are important to understand because of limitations in the threading model that OpenStack uses. OpenStack uses cooperative threading in python, which means that if something long and complicated is running, it will block other tasks inside that process from running unless it voluntarily yields execution to another cooperative thread.

A tangible example of this is the nova-compute process. In order to manage the image cache with libvirt, nova-compute has a periodic process which scans the contents of the image cache. Part of this scan is calculating a checksum for each of the images and making sure that checksum matches what nova-compute expects it to be. However, images can be very large and these checksums can take a long time to generate. At one point, before it was reported as a bug and fixed, nova-compute would block on this task and stop responding to RPC requests. This was visible to users as failure of operations such as spawning or deleting instances.

The take away from this is if you observe an OpenStack process which appears to "stop" for a while and then continue to process normally, you should check that periodic tasks aren't the problem. One way to do this is to disable the periodic tasks by setting their interval to zero. Additionally, you can configure how often these periodic tasks run -- in some cases it might make sense to run them at a different frequency from the default.

The frequency is defined separately for each periodic task. Therefore, to disable every periodic task in OpenStack Compute (Nova), you would need to set a number of configuration options to zero. The current list of configuration options you would need to set to zero are:

- bandwidth_poll_interval
- sync_power_state_interval
- heal_instance_info_cache_interval
- host_state_interval
- image_cache_manager_interval
- reclaim_instance_interval
- volume_usage_poll_interval

- shelved_poll_interval
- shelved_offload_time
- instance_delete_interval

To set a configuration option to zero, include a line such as `image_cache_manager_interval=0` to your `nova.conf` file.

This list will change between releases, so please refer to your configuration guide for up to date information.

Specific configuration topics

This section covers specific examples of configuration options you might consider tuning. It is by no means an exhaustive list.

OpenStack Compute (Nova)

Periodic task frequency

Before the Grizzly release, the frequency of periodic tasks was specified in seconds between runs. This meant that if the periodic task took 30 minutes to run and the frequency was set to hourly, then the periodic task actually ran every 90 minutes, because the task would wait an hour after running before running again. This changed in Grizzly, and we now time the frequency of periodic tasks from the start of the work the task does. So, our 30 minute periodic task will run every hour, with a 30 minute wait between the end of the first run and the start of the next.

APPENDIX A

Use Cases

This section contains a small selection of use cases from the community with more technical detail than usual. Further examples can be found on the OpenStack Website (https://www.openstack.org/user-stories/)

NeCTAR

Who uses it: Researchers from the Australian publicly funded research sector. Use is across a wide variety of disciplines, with the purpose of instances being from running simple web servers to using hundreds of cores for high throughput computing.

Deployment

Using OpenStack Compute Cells, the NeCTAR Cloud spans eight sites with approximately 4,000 cores per site.

Each site runs a different configuration, as resource cells in an OpenStack Compute cells setup. Some sites span multiple data centers, some use off compute node storage with a shared file system and some use on compute node storage with a non-shared file system. Each site deploys the Image Service with an Object Storage back-end. A central Identity Service, Dashboard and Compute API Service is used. Login to the Dashboard triggers a SAML login with Shibboleth, that creates an account in the Identity Service with an SQL back-end.

Compute nodes have 24 to 48 cores, with at least 4 GB of RAM per core and approximately 40 GB of ephemeral storage per core.

All sites are based on Ubuntu 12.04 with KVM as the hypervisor. The OpenStack version in use is typically the current stable version, with 5 to 10% back ported code from trunk and modifications.

Resources

- OpenStack.org Case Study (https://www.openstack.org/user-stories/nectar/)
- NeCTAR-RC GitHub (https://github.com/NeCTAR-RC/)
- NeCTAR Website (https://www.nectar.org.au/)

MIT CSAIL

Who uses it: Researchers from the MIT Computer Science and Artificial Intelligence Lab.

Deployment

The CSAIL cloud is currently 64 physical nodes with a total of 768 physical cores and 3,456 GB of RAM. Persistent data storage is largely outside of the cloud on NFS with cloud resources focused on compute resources. There are 65 users in 23 projects with typical capacity utilization nearing 90% we are looking to expand.

The software stack is Ubuntu 12.04 LTS with OpenStack Folsom from the Ubuntu Cloud Archive. KVM is the hypervisor, deployed using FAI (http://fai-project.org/) and Puppet for configuration management. The FAI and Puppet combination is used lab wide, not only for OpenStack. There is a single cloud controller node, with the remainder of the server hardware dedicated to compute nodes. Due to the compute intensive nature of the use case, the ratio of physical CPU and RAM to virtual is 1:1 in nova.conf. Although hyper-threading is enabled so, given the way Linux counts CPUs, this is actually 2:1 in practice.

On the network side, physical systems have two network interfaces and a separate management card for IPMI management. The OpenStack network service uses multi-host networking and the FlatDHCP.

DAIR

Who uses it: DAIR is an integrated virtual environment that leverages the CANARIE network to develop and test new information communication technology (ICT) and other digital technologies. It combines such digital infrastructure as advanced networking, and cloud computing and storage to create an environment for develop and test of innovative ICT applications, protocols and services, perform at-scale experimentation for deployment, and facilitate a faster time to market.

Deployment

DAIR is hosted at two different data centres across Canada: one in Alberta and the other in Quebec. It consists of a cloud controller at each location, however, one is designated as the "master" controller that is in charge of central authentication and quotas. This is done through custom scripts and light modifications to OpenStack. DAIR is currently running Folsom.

For Object Storage, each region has a Swift environment.

A NetApp appliance is used in each region for both block storage and instance storage. There are future plans to move the instances off of the NetApp appliance and onto a distributed file system such as Ceph or GlusterFS.

VlanManager is used extensively for network management. All servers have two bonded 10gb NICs that are connected to two redundant switches. DAIR is set up to use single-node networking where the cloud controller is the gateway for all instances on all compute nodes. Internal OpenStack traffic (for example, storage traffic) does not go through the cloud controller.

Resources

- DAIR Homepage (http://www.canarie.ca/en/dair-program/about)

CERN

Who uses it: Researchers at CERN (European Organization for Nuclear Research) conducting high-energy physics research.

Deployment

The environment is largely based on Scientific Linux 6, which is Red Hat compatible. We use KVM as our primary hypervisor although tests are ongoing with Hyper-V on Windows Server 2008.

We use the Puppet Labs OpenStack modules to configure Compute, Image Service, Identity Service and Dashboard. Puppet is used widely for instance configuration and Foreman as a GUI for reporting and instance provisioning.

Users and Groups are managed through Active Directory and imported into the Identity Service using LDAP. CLIs are available for Nova and Euca2ools to do this.

There are 3 clouds currently running at CERN, totaling around 3400 Nova Compute nodes, with approximately 60,000 cores. The CERN IT cloud aims to expand to 300,000 cores by 2015.

Resources

- OpenStack in Production: A tale of 3 OpenStack Clouds (openstack-in-production.blogspot.com/2013/09/a-tale-of-3-openstack-clouds-50000.html)
- Review of CERN Data Centre Infrastructure (http://cern.ch/go/N8wp)
- CERN Cloud Infrastructure User Guide (http://information-technology.web.cern.ch/book/cern-private-cloud-user-guide)

APPENDIX B

Tales From the Cryp^H^H^H^H Cloud

Herein lies a selection of tales from OpenStack cloud operators. Read, and learn from their wisdom.

Double VLAN

I was on-site in Kelowna, British Columbia, Canada setting up a new OpenStack cloud. The deployment was fully automated: Cobbler deployed the OS on the bare metal, bootstrapped it, and Puppet took over from there. I had run the deployment scenario so many times in practice and took for granted that everything was working.

On my last day in Kelowna, I was in a conference call from my hotel. In the background, I was fooling around on the new cloud. I launched an instance and logged in. Everything looked fine. Out of boredom, I ran *ps aux* and all of the sudden the instance locked up.

Thinking it was just a one-off issue, I terminated the instance and launched a new one. By then, the conference call ended and I was off to the data center.

At the data center, I was finishing up some tasks and remembered the lock-up. I logged into the new instance and ran *ps aux* again. It worked. Phew. I decided to run it one more time. It locked up. WTF.

After reproducing the problem several times, I came to the unfortunate conclusion that this cloud did indeed have a problem. Even worse, my time was up in Kelowna and I had to return back to Calgary.

Where do you even begin troubleshooting something like this? An instance just randomly locks when a command is issued. Is it the image? Nope — it happens on all images. Is it the compute node? Nope — all nodes. Is the instance locked up? No! New SSH connections work just fine!

We reached out for help. A networking engineer suggested it was an MTU issue. Great! MTU! Something to go on! What's MTU and why would it cause a problem?

MTU is maximum transmission unit. It specifies the maximum number of bytes that the interface accepts for each packet. If two interfaces have two different MTUs, bytes might get chopped off and weird things happen -- such as random session lockups.

Not all packets have a size of 1500. Running the *ls* command over SSH might only create a single packets less than 1500 bytes. However, running a command with heavy output, such as *ps aux* requires several packets of 1500 bytes.

OK, so where is the MTU issue coming from? Why haven't we seen this in any other deployment? What's new in this situation? Well, new data center, new uplink, new switches, new model of switches, new servers, first time using this model of servers… so, basically everything was new. Wonderful. We toyed around with raising the MTU at various areas: the switches, the NICs on the compute nodes, the virtual NICs in the instances, we even had the data center raise the MTU for our uplink interface. Some changes worked, some didn't. This line of troubleshooting didn't feel right, though. We shouldn't have to be changing the MTU in these areas.

As a last resort, our network admin (Alvaro) and myself sat down with four terminal windows, a pencil, and a piece of paper. In one window, we ran ping. In the second window, we ran *tcpdump* on the cloud controller. In the third, *tcpdump* on the compute node. And the forth had *tcpdump* on the instance. For background, this cloud was a multi-node, non-multi-host setup.

One cloud controller acted as a gateway to all compute nodes. VlanManager was used for the network config. This means that the cloud controller and all compute nodes had a different VLAN for each OpenStack project. We used the -s option of *ping* to change the packet size. We watched as sometimes packets would fully return, sometimes they'd only make it out and never back in, and sometimes the packets would stop at a random point. We changed *tcpdump* to start displaying the hex dump of the packet. We pinged between every combination of outside, controller, compute, and instance.

Finally, Alvaro noticed something. When a packet from the outside hits the cloud controller, it should not be configured with a VLAN. We verified this as true. When the packet went from the cloud controller to the compute node, it should only have a VLAN if it was destined for an instance. This was still true. When the ping reply was sent from the instance, it should be in a VLAN. True. When it came back to the cloud controller and on its way out to the public internet, it should no longer have a VLAN. False. Uh oh. It looked as though the VLAN part of the packet was not being removed.

That made no sense.

While bouncing this idea around in our heads, I was randomly typing commands on the compute node:

```
$ ip a
…
10: vlan100@vlan20: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc noqueue mas
ter br100 state UP
…
```

"Hey Alvaro, can you run a VLAN on top of a VLAN?"

"If you did, you'd add an extra 4 bytes to the packet…"

Then it all made sense…

```
$ grep vlan_interface /etc/nova/nova.conf
vlan_interface=vlan20
```

In `nova.conf`, `vlan_interface` specifies what interface OpenStack should attach all VLANs to. The correct setting should have been:

```
vlan_interface=bond0
```

.

As this would be the server's bonded NIC.

vlan20 is the VLAN that the data center gave us for outgoing public internet access. It's a correct VLAN and is also attached to bond0.

By mistake, I configured OpenStack to attach all tenant VLANs to vlan20 instead of bond0 thereby stacking one VLAN on top of another which then added an extra 4 bytes to each packet which cause a packet of 1504 bytes to be sent out which would cause problems when it arrived at an interface that only accepted 1500!

As soon as this setting was fixed, everything worked.

"The Issue"

At the end of August 2012, a post-secondary school in Alberta, Canada migrated its infrastructure to an OpenStack cloud. As luck would have it, within the first day or two of it running, one of their servers just disappeared from the network. Blip. Gone.

After restarting the instance, everything was back up and running. We reviewed the logs and saw that at some point, network communication stopped and then everything went idle. We chalked this up to a random occurrence.

A few nights later, it happened again.

We reviewed both sets of logs. The one thing that stood out the most was DHCP. At the time, OpenStack, by default, set DHCP leases for one minute (it's now two minutes). This means that every instance contacts the cloud controller (DHCP server) to renew its fixed IP. For some reason, this instance could not renew its IP. We correlated the instance's logs with the logs on the cloud controller and put together a conversation:

1. Instance tries to renew IP.
2. Cloud controller receives the renewal request and sends a response.
3. Instance "ignores" the response and re-sends the renewal request.
4. Cloud controller receives the second request and sends a new response.
5. Instance begins sending a renewal request to `255.255.255.255` since it hasn't heard back from the cloud controller.
6. The cloud controller receives the `255.255.255.255` request and sends a third response.
7. The instance finally gives up.

With this information in hand, we were sure that the problem had to do with DHCP. We thought that for some reason, the instance wasn't getting a new IP address and with no IP, it shut itself off from the network.

A quick Google search turned up this: DHCP lease errors in VLAN mode (https://lists.launchpad.net/openstack/msg11696.html) which further supported our DHCP theory.

An initial idea was to just increase the lease time. If the instance only renewed once every week, the chances of this problem happening would be tremendously smaller than every minute. This didn't solve the problem, though. It was just covering the problem up.

We decided to have *tcpdump* run on this instance and see if we could catch it in action again. Sure enough, we did.

The *tcpdump* looked very, very weird. In short, it looked as though network communication stopped before the instance tried to renew its IP. Since there is so much DHCP chatter from a one minute lease, it's very hard to confirm it, but even with only milliseconds difference between packets, if one packet arrives first, it arrived first, and if that packet reported network issues, then it had to have happened before DHCP.

Additionally, this instance in question was responsible for a very, very large backup job each night. While "The Issue" (as we were now calling it) didn't happen exactly

when the backup happened, it was close enough (a few hours) that we couldn't ignore it.

Further days go by and we catch The Issue in action more and more. We find that dhclient is not running after The Issue happens. Now we're back to thinking it's a DHCP issue. Running `/etc/init.d/networking` restart brings everything back up and running.

Ever have one of those days where all of the sudden you get the Google results you were looking for? Well, that's what happened here. I was looking for information on dhclient and why it dies when it can't renew its lease and all of the sudden I found a bunch of OpenStack and dnsmasq discussions that were identical to the problem we were seeing!

Problem with Heavy Network IO and Dnsmasq (http://www.gossamer-threads.com/lists/openstack/operators/18197)

instances losing IP address while running, due to No DHCPOFFER (http://www.gossamer-threads.com/lists/openstack/dev/14696)

Seriously, Google.

This bug report was the key to everything: KVM images lose connectivity with bridged network (https://bugs.launchpad.net/ubuntu/+source/qemu-kvm/+bug/997978)

It was funny to read the report. It was full of people who had some strange network problem but didn't quite explain it in the same way.

So it was a qemu/kvm bug.

At the same time of finding the bug report, a co-worker was able to successfully reproduce The Issue! How? He used *iperf* to spew a ton of bandwidth at an instance. Within 30 minutes, the instance just disappeared from the network.

Armed with a patched qemu and a way to reproduce, we set out to see if we've finally solved The Issue. After 48 hours straight of hammering the instance with bandwidth, we were confident. The rest is history. You can search the bug report for "joe" to find my comments and actual tests.

Disappearing Images

At the end of 2012, Cybera (a nonprofit with a mandate to oversee the development of cyberinfrastructure in Alberta, Canada) deployed an updated OpenStack cloud for their DAIR project (http://www.canarie.ca/en/dair-program/about). A few days into production, a compute node locks up. Upon rebooting the node, I checked to see what instances were hosted on that node so I could boot them on behalf of the customer. Luckily, only one instance.

The *nova reboot* command wasn't working, so I used *virsh*, but it immediately came back with an error saying it was unable to find the backing disk. In this case, the backing disk is the Glance image that is copied to `/var/lib/nova/instances/_base` when the image is used for the first time. Why couldn't it find it? I checked the directory and sure enough it was gone.

I reviewed the `nova` database and saw the instance's entry in the `nova.instances` table. The image that the instance was using matched what virsh was reporting, so no inconsistency there.

I checked Glance and noticed that this image was a snapshot that the user created. At least that was good news — this user would have been the only user affected.

Finally, I checked StackTach and reviewed the user's events. They had created and deleted several snapshots — most likely experimenting. Although the timestamps didn't match up, my conclusion was that they launched their instance and then deleted the snapshot and it was somehow removed from `/var/lib/nova/instances/_base`. None of that made sense, but it was the best I could come up with.

It turns out the reason that this compute node locked up was a hardware issue. We removed it from the DAIR cloud and called Dell to have it serviced. Dell arrived and began working. Somehow or another (or a fat finger), a different compute node was bumped and rebooted. Great.

When this node fully booted, I ran through the same scenario of seeing what instances were running so I could turn them back on. There were a total of four. Three booted and one gave an error. It was the same error as before: unable to find the backing disk. Seriously, what?

Again, it turns out that the image was a snapshot. The three other instances that successfully started were standard cloud images. Was it a problem with snapshots? That didn't make sense.

A note about DAIR's architecture: `/var/lib/nova/instances` is a shared NFS mount. This means that all compute nodes have access to it, which includes the `_base` directory. Another centralized area is `/var/log/rsyslog` on the cloud controller. This directory collects all OpenStack logs from all compute nodes. I wondered if there were any entries for the file that *virsh* is reporting:

```
dair-ua-c03/nova.log:Dec 19 12:10:59 dair-ua-c03
2012-12-19 12:10:59 INFO nova.virt.libvirt.imagecache
[-] Removing base file:
/var/lib/nova/instances/_base/7b4783508212f5d242cbf9ff56fb8d33b4ce6166_10
```

Ah-hah! So OpenStack was deleting it. But why?

A feature was introduced in Essex to periodically check and see if there were any `_base` files not in use. If there were, Nova would delete them. This idea sounds innocent enough and has some good qualities to it. But how did this feature end up turned on? It was disabled by default in Essex. As it should be. It was decided to be turned on in Folsom (https://bugs.launchpad.net/nova/+bug/1029674). I cannot emphasize enough that:

Actions which delete things should not be enabled by default.

Disk space is cheap these days. Data recovery is not.

Secondly, DAIR's shared `/var/lib/nova/instances` directory contributed to the problem. Since all compute nodes have access to this directory, all compute nodes periodically review the _base directory. If there is only one instance using an image, and the node that the instance is on is down for a few minutes, it won't be able to mark the image as still in use. Therefore, the image seems like it's not in use and is deleted. When the compute node comes back online, the instance hosted on that node is unable to start.

The Valentine's Day Compute Node Massacre

Although the title of this story is much more dramatic than the actual event, I don't think, or hope, that I'll have the opportunity to use "Valentine's Day Massacre" again in a title.

This past Valentine's Day, I received an alert that a compute node was no longer available in the cloud — meaning,

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

showed this particular node with a status of `XXX`

.

I logged into the cloud controller and was able to both *ping* and SSH into the problematic compute node which seemed very odd. Usually if I receive this type of alert, the compute node has totally locked up and would be inaccessible.

After a few minutes of troubleshooting, I saw the following details:

- A user recently tried launching a CentOS instance on that node
- This user was the only user on the node (new node)
- The load shot up to 8 right before I received the alert

- The bonded 10gb network device (bond0) was in a DOWN state
- The 1gb NIC was still alive and active

I looked at the status of both NICs in the bonded pair and saw that neither was able to communicate with the switch port. Seeing as how each NIC in the bond is connected to a separate switch, I thought that the chance of a switch port dying on each switch at the same time was quite improbable. I concluded that the 10gb dual port NIC had died and needed replaced. I created a ticket for the hardware support department at the data center where the node was hosted. I felt lucky that this was a new node and no one else was hosted on it yet.

An hour later I received the same alert, but for another compute node. Crap. OK, now there's definitely a problem going on. Just like the original node, I was able to log in by SSH. The bond0 NIC was DOWN but the 1gb NIC was active.

And the best part: the same user had just tried creating a CentOS instance. What?

I was totally confused at this point, so I texted our network admin to see if he was available to help. He logged in to both switches and immediately saw the problem: the switches detected spanning tree packets coming from the two compute nodes and immediately shut the ports down to prevent spanning tree loops:

```
       Feb 15 01:40:18 SW-1 Stp: %SPANTREE-4-BLOCK_BPDUGUARD: Received BPDU pack
et on Port-Channel35 with BPDU guard enabled. Disabling interface. (source mac
fa:16:3e:24:e7:22)
Feb 15 01:40:18 SW-1 Ebra: %ETH-4-ERRDISABLE: bpduguard error detected on Port-
Channel35.
Feb 15 01:40:18 SW-1 Mlag: %MLAG-4-INTF_INACTIVE_LOCAL: Local interface Port-
Channel35 is link down. MLAG 35 is inactive.
Feb 15 01:40:18 SW-1 Ebra: %LINEPROTO-5-UPDOWN: Line protocol on Interface Port-
Channel35 (Server35), changed state to down
Feb 15 01:40:19 SW-1 Stp: %SPANTREE-6-INTERFACE_DEL: Interface Port-Channel35
has been removed from instance MST0
Feb 15 01:40:19 SW-1 Ebra: %LINEPROTO-5-UPDOWN: Line protocol on Interface Ether
net35 (Server35), changed state to down
```

He re-enabled the switch ports and the two compute nodes immediately came back to life.

Unfortunately, this story has an open ending... we're still looking into why the CentOS image was sending out spanning tree packets. Further, we're researching a proper way on how to mitigate this from happening. It's a bigger issue than one might think. While it's extremely important for switches to prevent spanning tree loops, it's very problematic to have an entire compute node be cut from the network when this happens. If a compute node is hosting 100 instances and one of them sends a spanning tree packet, that instance has effectively DDOS'd the other 99 instances.

This is an ongoing and hot topic in networking circles — especially with the raise of virtualization and virtual switches.

Down the Rabbit Hole

Users being able to retrieve console logs from running instances is a boon for support — many times they can figure out what's going on inside their instance and fix what's going on without bothering you. Unfortunately, sometimes overzealous logging of failures can cause problems of its own.

A report came in: VMs were launching slowly, or not at all. Cue the standard checks — nothing on the nagios, but there was a spike in network towards the current master of our RabbitMQ cluster. Investigation started, but soon the other parts of the queue cluster were leaking memory like a sieve. Then the alert came in — the master rabbit server went down. Connections failed over to the slave.

At that time, our control services were hosted by another team and we didn't have much debugging information to determine what was going on with the master, and couldn't reboot it. That team noted that it failed without alert, but managed to reboot it. After an hour, the cluster had returned to its normal state and we went home for the day.

Continuing the diagnosis the next morning was kick started by another identical failure. We quickly got the message queue running again, and tried to work out why Rabbit was suffering from so much network traffic. Enabling debug logging on `nova-api` quickly brought understanding. A *tail -f /var/log/nova/nova-api.log* was scrolling by faster than we'd ever seen before. CTRL+C on that and we could plainly see the contents of a system log spewing failures over and over again - a system log from one of our users' instances.

After finding the instance ID we headed over to `/var/lib/nova/instances` to find the `console.log`:

```
adm@cc12:/var/lib/nova/instances/instance-00000e05# wc -l console.log
92890453 console.log
adm@cc12:/var/lib/nova/instances/instance-00000e05# ls -sh console.log
5.5G console.log
```

Sure enough, the user had been periodically refreshing the console log page on the dashboard and the 5G file was traversing the rabbit cluster to get to the dashboard.

We called them and asked them to stop for a while, and they were happy to abandon the horribly broken VM. After that, we started monitoring the size of console logs.

To this day, the issue (https://bugs.launchpad.net/nova/+bug/832507) doesn't have a permanent resolution, but we look forward to the discussion at the next summit.

APPENDIX C
Resources

OpenStack

OpenStack Configuration Reference (http://docs.openstack.org/trunk/config-reference/content/section_compute-hypervisors.html)

OpenStack Install Guide - Ubuntu (http://docs.openstack.org/havana/install-guide/install/apt/content/)

OpenStack Cloud Administrator Guide (http://docs.openstack.org/admin-guide-cloud/content/)

OpenStack Security Guide (http://docs.openstack.org/security-guide/content/)

OpenStack Cloud Computing Cookbook (http://www.packtpub.com/openstack-cloud-computing-cookbook-second-edition/book)

Cloud (general)

NIST Cloud Computing Definition (http://csrc.nist.gov/publications/nistpubs/800-145/SP800-145.pdf)

Python

Dive Into Python (http://www.diveintopython.net)

Networking

TCP/IP Illustrated (http://www.pearsonhighered.com/educator/product/TCPIP-Illustrated-Volume-1-The-Protocols/9780321336316.page)

The TCP/IP Guide (http://nostarch.com/tcpip.htm)

A tcpdump Tutorial and Primer (http://danielmiessler.com/study/tcpdump/)

Systems administration

UNIX and Linux Systems Administration Handbook (http://www.admin.com/)

Virtualization

The Book of Xen (http://nostarch.com/xen.htm)

Configuration management

Puppet Labs Documentation (http://docs.puppetlabs.com/)

Pro Puppet (http://www.apress.com/9781430230571)

Glossary

Use this glossary to get definitions of OpenStack-related words and phrases.

To add to this glossary, fork the `openstack/openstack-manuals` repository on github.com and update the source files through the OpenStack contribution process.

account
: The swift context of an account, or a user account from an identity service such as Active Directory, /etc/passwd, OpenLDAP, keystone, and so on.

account auditor
: Checks for missing replicas, incorrect, and corrupted objects in a specified swift account by running queries against the back-end SQLite database.

account database
: An SQLite database that contains swift accounts and related metadata and is accessed by the accounts server. Alternately, the keystone back-end which contains accounts.

account reaper
: A swift worker that scans for and deletes account databases that are marked for deletion on an account server.

account server
: Lists containers in swift and stores container information in the account database.

account service
: Component of swift that provides account services such as list, create, modify, and audit. Do not confuse with keystone, OpenLDAP, or similar user account services.

Active Directory
: Authentication and Identity Service by Microsoft, based on LDAP. Supported in OpenStack.

address pool
: A group of fixed and/or floating IP addresses that are assigned to a nova project and can be used by or assigned to the VM instances in a project.

admin API
: A subset of API calls that are accessible to authorized administrators and are generally not accessible to end users or the public internet, can exist as a separate service (keystone) or can be a subset of another API (nova).

Amazon Kernel Image (AKI)
: Both a VM container format and a VM disk format. Supported by glance.

Amazon Machine Image (AMI)
: Both a VM container format and a VM disk format. Supported by glance.

Amazon Ramdisk Image (ARI)
: Both a VM container format and a VM disk format. Supported by glance.

Apache
: The most common web server software currently used on the Internet, known as HTTPd.

Apache License 2.0
: All OpenStack core projects are provided under the terms of the Apache License 2.0 license.

API endpoint
: The daemon, worker, or service that a client communicates with to access an API. In OpenStack, API endpoints can provide services such as authentication, adding images, booting virtual machines, and attaching volumes.

API extension
: A feature of nova and quantum that allows custom modules to extend the core APIs.

API extension plug-in
: Alternative term for a quantum plug-in or quantum API extension.

API server
: Any node running a daemon or worker that provides an API endpoint.

API version
: In OpenStack, a the API version for a project is part of the URL. For example, `example.com/nova/v1/foobar`.

Application Programming Interface (API)
: A collection of specifications used to access a service, application, or program. Includes service calls, required parameters for each call, and the expected return values.

arptables
: Used along with iptables, ebtables, and ip6tables in nova to provide firewall services.

Asynchronous JavaScript and XML (AJAX)
: A group of interrelated web development techniques used on the client-side to create asynchronous web applications. Used extensively in horizon.

attachment (network)
: Association of an interface ID to a logical port. Plugs an interface into a port.

auditor
: A worker process that verifies the integrity of swift objects, containers, and accounts. Auditors is the collective term for the swift account auditor, container auditor, and object auditor.

Austin
: Project name for the initial release of OpenStack.

authentication
: The process that confirms that the user, process, or client is really who they say they are through private key, secret token, password, fingerprint, or similar method. Abbreviated as AuthN.

authentication token
: A string of text provided to the client after authentication. Must be provided by the user or process in subsequent requests to the API endpoint.

authorization
: The act of verifying that a user, process, or client is authorized to perform an action, such as delete a swift object, list a swift container, start a nova VM, reset a password, and so on. Abbreviate as AuthZ.

availability zone
: A segregated area of a cloud deployment.

back-end catalog
The storage method used by the keystone catalog service to store and retrieve information about API endpoints that are available to the client. Examples include a SQL database, LDAP database, or KVS back-end.

back-end store
The persistent data store used that glance uses to retrieve and store VM images. Options include swift, local file system, S3, and HTTP.

bare
A glance container format that indicates that no container exists for the VM image.

Bexar
A grouped release of projects related to OpenStack that came out in February of 2011. It included Compute (nova) and Object Storage (swift) only.

block device
A device that moves data in the form of blocks. These device nodes interface the devices, such as hard disks, CD-ROM drives, flash drives, and other addressable regions of memory.

block migration
A method of VM live migration used by KVM to evacuate instances from one host to another with very little downtime during a user-initiated switch-over. Does not require shared storage. Supported by nova.

bootable disk image
A type of VM image that exists as a single, bootable file.

builder file
Contains configuration information for a swift ring, and is used to re-configure the ring or to recreate it from scratch after a serious failure.

cache pruner
An executable program that is used to keep a glance VM image cache at or below its configured maximum size.

Cactus
An OpenStack grouped release of projects that came out in the spring of 2011. It included Compute (nova), Object Storage (swift), and the Image service (glance).

capability
Defines resources for a cell, including CPU, storage, and networking. Can apply to the specific services within a cell or a whole cell.

capacity cache
A table within the nova back-end database that contains the current workload, amount of free RAM, number of VMs running on each host. Used to determine on which VM a host starts.

capacity updater
A notification driver that monitors VM instances and updates the capacity cache as needed.

catalog
Contains a list of available API endpoints to a user after they authenticate to keystone.

catalog service
A keystone service that provides a list of available API endpoints to a user after they authenticate to keystone.

ceilometer
An incubated project that provides metering and billing facilities for OpenStack.

cell
Provides logical partitioning of nova resources in a child and parent relationship. Requests are passed from parent cells to child cells if the parent cannot provide the requested resource.

cell forwarding
: A nova option that allows parent cells to pass resource requests to child cells if the parent cannot provide the requested resource.

cell manager
: The nova component that contains a list of the current capabilities of each host within the cell and routes requests as appropriate.

Ceph
: Massively scalable distributed storage system that consists of an object store, block store, and POSIX-compatible distributed file system. Compatible with OpenStack.

CephFS
: The POSIX-compliant file system provided by Ceph.

certificate authority
: A simple certificate authority provided by nova for cloudpipe VPNs and VM image decryption.

chance scheduler
: A scheduling method used by nova that randomly chooses an available host from the pool.

changes-since
: A nova API parameter that allows you to download changes to the requested item since your last request, instead of downloading a new, fresh set of data and comparing it against the old data.

Chef
: A configuration management tool that supports OpenStack.

child cell
: If a requested resource such as CPU time, disk storage, or memory is not available in the parent cell, the request is forwarded to its associated child cells. If the child cell can fulfill the request, it does. Otherwise, it attempts to pass the request to any of its children.

cinder
: The OpenStack Block Storage service that maintains the block devices that can be attached to virtual machine instances.

cloud architect
: A person who plans, designs, and oversees the creation of clouds.

cloud controller node
: A node that runs network, volume, API, scheduler and image services. Each service may be broken out into separate nodes for scalability or availability.

cloud-init
: A package commonly installed in VM images that performs initialization of an instance after boot using information that it retrieves from the metadata service such as the SSH public key and user data.

cloudpipe
: A service in nova used to create VPNs on a per-project basis.

cloudpipe image
: A pre-made VM image that serves as a cloudpipe server. Essentially, OpenVPN running on Linux.

command filter
: Lists allowed commands within the nova rootwrap facility.

community project
: A project that is not officially endorsed by the OpenStack Foundation. If the project is successful enough, it might be elevated to an incubated project and then to a core project, or it might be merged with the main code trunk.

Compute API
: The nova-api daemon that provides access to the nova services. Can also communicate with some outside APIs such as the Amazons EC2 API.

Compute API extension
: Alternative term for a nova API extension.

compute controller
: The nova component that chooses suitable hosts on which to start VM instances.

compute node
: A node that runs the nova-compute daemon and the virtual machine instances.

compute service
: Alternative term for the nova component that manages VMs.

concatenated object
: A segmented large object within swift that is put back together again and then sent to the client.

consistency window
: The amount of time it takes for a new swift object to become accessible to all clients.

console log
: Contains the output from a Linux VM console in nova.

container
: Used to organize and store objects within swift, similar to the concept as a Linux directory but cannot be nested. Alternative term for a glance container format.

container auditor
: Checks for missing replicas or incorrect objects in the specified swift containers through queries to the SQLite back-end database.

container database
: A SQLite database that contains swift containers and related metadata and is accessed by the container server

container format
: The "envelope" used by glance to store a VM image and its associated metadata, such as machine state, OS disk size, and so on.

container server
: Component of swift that manages containers.

container service
: The swift component that provides container services, such as create, delete, list, and so on.

controller node
: Alternative term for a cloud controller node.

core API
: Depending on context, the core API is either the OpenStack API or the main API of a specific core project, such as nova, quantum, glance, and so on.

core project
: An official OpenStack project. Currently consists of Compute (nova), Object Storage (swift), Image Service (glance), Identity (keystone), Dashboard (horizon), Networking (quantum), and Volume (cinder).

credentials
: Data that is only known to or accessible by a user that is used to verify the user is who they say they are and presented to the server during authentication. Examples include a password, secret key, digital certificate, fingerprint, and so on.

Crowbar
: An open source community project by Dell that aims to provide all necessary services to quickly deploy clouds.

current workload
: An element of the nova capacity cache that is calculated based on the number of build, snapshot, migrate, and resize operations currently in progress on a given host.

customization module
: A user-created Python module that is loaded by horizon to change the look and feel of the dashboard.

dashboard
The web-based management interface for OpenStack. An alternative name for horizon.

database replicator
The component of swift that copies changes in the account, container, and object databases to other nodes.

default panel
The panel that is displayed when a user accesses the horizon dashboard.

default tenant
New users are assigned to this keystone tenant if no tenant is specified when a user is created.

default token
A keystone token that is not associated with a specific tenant and is exchanged for a scoped token.

delayed delete
An option within glance so that rather than immediately delete an image, it is deleted after a pre-defined number of seconds.

delivery mode
Setting for the nova RabbitMQ message delivery mode, can be set to either transient or persistent.

device
In the context of swift this refers to the underlying storage device.

device ID
Maps swift partitions to physical storage devices.

device weight
Used to distribute the partitions among swift devices. The distribution is usually proportional to the storage capacity of the device.

ebtables
Used in nova along with arptables, iptables, and ip6tables to create firewalls and to ensure isolation of network communications.

DevStack
Community project that uses shell scripts to quickly deploy complete OpenStack development environments.

Diablo
A grouped release of projects related to OpenStack that came out in the fall of 2011, the fourth release of OpenStack. It included Compute (nova 2011.3), Object Storage (swift 1.4.3), and the Image service (glance).

disk format
The underlying format that a disk image for a VM is stored as within the glance back-end store. For example, AMI, ISO, QCOW2, VMDK, and so on.

dispersion
In swift, tools to test and ensure dispersion of objects and containers to ensure fault tolerance.

Django
A web framework used extensively in horizon.

dnsmasq
Daemon that provides DNS, DHCP, BOOTP, and TFTP services, used by the nova VLAN manager and FlatDHCP manager.

DNS record
A record that specifies information about a particular domain and belongs to the domain.

Dynamic Host Configuration Protocol (DHCP)
A method to automatically configure networking for a host at boot time. Provided by both quantum and nova.

EC2
: The Amazon Elastic Compute Cloud, a public cloud run by Amazon that provides similar functionality to nova.

EC2 access key
: Used along with an EC2 secret key to access the nova EC2 API.

EC2 API
: OpenStack supports accessing the Amazon EC2 API through nova.

EC2 Compatibility API
: A nova component that allows OpenStack to communicate with Amazon EC2

EC2 secret key
: Used along with an EC2 access key when communicating with the nova EC2 API, is used to digitally sign each request.

Elastic Block Storage (EBS)
: The Amazon commercial block storage product, similar to cinder.

endpoint
: See API endpoint.

endpoint registry
: Alternative term for a keystone catalog.

endpoint template
: A list of URL and port number endpoints that indicate where a service, such as object storage, compute, identity, and so on, can be accessed.

entity
: Any piece of hardware or software that wants to connect to the network services provided by quantum, the Network Connectivity service. An entity can make use of quantum by implementing a VIF.

ephemeral storage
: A storage volume attached to a virtual machine instance that does not persist after the instance is terminated.

Essex
: A grouped release of projects related to OpenStack that came out in April 2012, the fifth release of OpenStack. It included Compute (nova 2012.1), Object Storage (swift 1.4.8), Image (glance), Identity (keystone), and Dashboard (horizon).

ESX
: An OpenStack-supported hypervisor, owned by VMware.

ESXi
: An OpenStack-supported hypervisor, owned by VMware.

ETag
: MD5 hash of an object within swift, used to ensure data integrity.

euca2ools
: A collection of command line tools for administering VMs, most are compatible with OpenStack.

evacuate
: The process of migrating one or all virtual machine (VM) instances from one host to another, compatible with both shared storage live migration and block migration.

extension
: Alternative term for a nova API extension or plug-in. In the context of keystone this is a call that is specific to the implementation, such as adding support for OpenID.

extra specs
: Additional requirements that a user can specify when requesting a new instance, examples include a minimum amount of network bandwidth or a GPU.

FakeLDAP
An easy method to create a local LDAP directory for testing keystone and nova. Requires Redis.

fill-first scheduler
The nova scheduling method that attempts to fill a host with VMs rather than starting new VMs on a variety of hosts.

filter
The step of the nova scheduling process where hosts that cannot run the VMs are eliminated and are not chosen.

firewall
Used to restrict communications between hosts and/or nodes, implemented in nova using iptables, arptables, ip6tables and etables.

Fixed IP address
An IP address that is associated with the same instance each time that instance boots, generally not accessible to end users or the public internet, used for management of the instance.

FlatDHCP Manager
A nova networking manager that provides a single Layer 2 domain for all subnets in the OpenStack cloud. Provides a single DHCP server for each instance of nova-network to assign and manage IP addresses for all instances.

Flat Manager
The nova component that gives IP addresses to authorized nodes and assumes DHCP, DNS, and routing configuration and services are provided by something else.

flat mode injection
A nova networking method where the OS network configuration information is injected into the VM (VM) image before the instance starts.

flat network
A nova network configuration where all of the instances have IP addresses on the same subnet. Flat networks do not use VLANs.

flavor
Describes the parameters of the various virtual machine images that are available to users, includes parameters such as CPU, storage, and memory. Also known as instance type.

flavor ID
UUID for each nova or glance VM flavor or instance type.

Floating IP address
An IP address that a nova project can associate with a VM so the instance has the same public IP address each time that it boots. You create a pool of floating IP addresses and assign them to instances as they are launched to maintain a consistent IP address for maintaining DNS assignment.

Folsom
A grouped release of projects related to OpenStack that came out in the fall of 2012, the sixth release of OpenStack. It includes Compute (nova), Object Storage (swift), Identity (keystone), Networking (quantum), Image service (glance) and Volumes or Block Storage (cinder).

FormPost
swift middleware that allows users to upload (post) an image through a form on a web page.

glance
A core project that provides the OpenStack Image Service.

glance API server
Processes client requests for VMs, updates glance metadata on the registry server, and communicates with the store adapter

to upload VM images from the back-end store.

global endpoint template
The keystone endpoint template that contains services available to all tenants.

GlusterFS
An open-source, distributed, shared file system,

handover
An object state in swift where a new replica of the object is automatically created due to a drive failure.

hard reboot
A type of reboot where a physical or virtual power button is pressed as opposed to a graceful, proper shutdown of the operating system.

Heat
An integrated project that aims to orchestrate multiple cloud applications for OpenStack.

horizon
The project that provides the OpenStack Dashboard.

host
A physical computer, also known as a node. Contrast with: instance.

ID number
Unique numeric ID associated with each user in keystone, conceptually similar to a Linux or LDAP UID.

Identity API
Alternative term for the Identity Service API.

Identity back-end
The source used by keystone to retrieve user information an OpenLDAP server for example.

Identity Service
Provides authentication services, also known as keystone.

Grizzly
Project name for the seventh release of OpenStack.

guest OS
An operating system instance running under the control of a hypervisor.

host aggregate
A method to further subdivide availability zones into a collection of hosts.

Hyper-V
One of the hypervisors supported by OpenStack, developed by Microsoft.

hypervisor
Software that arbitrates and controls VM access to the actual underlying hardware.

hypervisor pool
A collection of hypervisors grouped together through host aggregates.

Identity Service API
The API used to access the OpenStack Identity Service provided through keystone.

image
A collection of files for a specific operating system (OS) that you use to create or rebuild a server. You can also create custom images, or snapshots, from servers that you have launched.

Image API
The glance API endpoint for management of VM images.

image cache
: Used by glance to allow images on the local host to be used rather than redownloading them from the image server each time one is requested.

image ID
: Combination of URI and UUID used to access glance VM images through the image API.

image membership
: A list of tenants that can access a given VM image within glance.

image owner
: The keystone tenant who owns a glance virtual machine image.

image registry
: A list of VM images that are available through glance.

Image Service API
: Alternative name for the glance image API.

image status
: The current status of a VM image in glance, not to be confused with the status of a running instance.

image store
: The back-end store used by glance to store VM images, options include swift, local file system, S3, or HTTP.

image UUID
: The UUID used by glance to uniquely identify each VM image.

incubated project
: A community project may be elevated to this status and is then promoted to a core project.

JavaScript Object Notation (JSON)
: One of the supported response formats for the OpenStack API.

Jenkins
: Tool used for OpenStack development to run jobs automatically.

ingress filtering
: The process of filtering incoming network traffic. Supported by nova.

injection
: The process of putting a file into a virtual machine image before the instance is started.

instance
: A running VM, or a VM in a known state such as suspended that can be used like a hardware server.

instance ID
: Unique ID that is specific to each running nova VM instance.

instance state
: The current state of a nova VM image.

instance type
: Alternative term for flavor.

instance type ID
: Alternative term for a flavor ID.

instance UUID
: Unique ID assigned to each nova VM instance.

interface ID
: Unique ID for a quantum VIF or vNIC in the form of a UUID.

ip6tables
: Used along with arptables, ebtables, and iptables to create firewalls in nova.

iptables
: Used along with arptables, ebtables, and ip6tables to create firewalls in nova.

kernel-based VM (KVM)
An OpenStack-supported hypervisor

keystone
The project that provides OpenStack Identity services.

large object
An object within swift that is larger than 5 GBs.

Launchpad
The collaboration site for OpenStack.

Layer-2 network
Term used for OSI network architecture for the data link layer.

libvirt
Virtualization API library used by OpenStack to interact with many of its supported hypervisors, including KVM, QEMU and LXC.

Linux bridge
Software used to allow multiple VMs to share a single physical NIC within nova.

management API
Alternative term for an admin API.

management network
A network segment used for administration, not accessible to the public internet.

manifest
Used to track segments of a large object within swift.

manifest object
A special swift object that contains the manifest for a large object.

membership
The association between a glance VM image and a tenant, allows images to be shared with specified tenant(s).

membership list
Contains a list of tenants that can access a given VM image within glance.

Kickstart
A tool to automate system configuration and installation on Red Hat, Fedora, and CentOS based Linux distributions.

Linux bridge quantum plug-in
Plugin that allows a Linux bridge to understand a quantum port, interface attachment, and other abstractions.

Linux containers (LXC)
An OpenStack-supported hypervisor.

live migration
The ability within nova to move running virtual machine instances from one host to another with only a small service interruption during switch-over.

memory overcommit
The ability to start new VM instances based on the actual memory usage of a host, as opposed to basing the decision on the amount of RAM each running instance thinks it has available. Also known as RAM overcommit.

message broker
The software package used to provide AMQP messaging capabilities within nova, default is RabbitMQ.

message bus
The main virtual communication line used by all AMQP messages for intercloud communications within nova.

message queue
Passes requests from clients to the appropriate workers and returns the output to the client once the job is complete.

migration
The process of moving a VM instance from one host to another.

multinic
Facility in nova that allows each virtual machine instance to have more than one VIF connected to it.

network ID
Unique ID assigned to each network segment within quantum.

network manager
The nova component that manages various network components, such as firewall rules, IP address allocation, and so on.

network node
Any nova node that runs the network worker daemon.

network segment
Represents a virtual, isolated OSI layer 2 subnet in quantum.

network UUID
Unique ID for a quantum network segment.

network worker
The nova-network worker daemon, provides services such as giving an IP address to a booting nova instance.

non-persistent volume
Alternative term for an ephemeral volume.

nova
The OpenStack project that provides compute services.

nova API
Alternative term for the nova Compute API.

nova-network
A nova component that manages IP address allocation, firewalls, and other network-related tasks.

object
A BLOB of data held by swift, can be in any format.

Object API
Alternative term for the swift object API.

object auditor
Opens all objects for an object server and verifies the MD5 hash, size, and metadata for each object.

object expiration
A configurable option within swift to automatically delete objects after a specified amount of time has passed or a certain date is reached.

object hash
Uniquely ID for a swift object.

object path hash
Used by swift to determine the location of an object in the ring. Maps objects to partitions.

object replicator
Component of swift that copies and object to remote partitions for fault tolerance.

object server
Component of swift that is responsible for managing objects.

Object Service API
Alternative term for the swift object API.

object storage
Provides eventually consistent and redundant storage and retrieval of fixed digital content.

object versioning
Allows a user to set a flag on a swift container so all objects within the container are versioned.

parent cell
If a requested resource, such as CPU time, disk storage, or memory, is not available in the parent cell, the request is forwarded to associated child cells.

partition
A unit of storage within swift used to store objects, exists on top of devices, replicated for fault tolerance.

partition index
Contains the locations of all swift partitions within the ring.

partition shift value
Used by swift to determine which partition data should reside on.

pause
A VM state where no changes occur (no changes in memory, network communications stop, etc), the VM is frozen but not shut down.

persistent volume
Disk volumes that persist beyond the lifetime of individual virtual machine instances. Contrast with: ephemeral storage

plugin
Software component providing the actual implementation for quantum APIs, or for Compute APIs, depending on the context.

policy service
Component of keystone that provides a rule management interface and a rule based authorization engine.

port
A virtual network port within quantum, VIFs / vNICs are connected to a port.

operator
The person responsible for planning and maintaining an OpenStack installation.

port UUID
Unique ID for a quantum port.

preseed
A tool to automate system configuration and installation on Debian based Linux distributions.

private image
A glance VM image that is only available to specified tenants.

project
A logical grouping of users within nova, used to define quotas and access to VM images.

project ID
User defined alpha-numeric string in nova, the name of a project.

project VPN
Alternative term for a cloudpipe.

proxy node
A node that provides the swift proxy service.

proxy server
Users of swift interact with the service through the proxy server which in-turn looks up the location of the requested data within the ring and returns the results to the user.

public API
An API endpoint used for both service to service communication and end user interactions.

public image
A glance VM image that is available to all tenants.

public IP address
An IP address that is accessible to end-users.

public network
The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. The public network interface is controlled by the public_interface option.

Puppet
A configuration management tool that supports OpenStack.

Python
Programming language used extensively in OpenStack.

quantum
A core OpenStack project that provides a network connectivity abstraction layer to OpenStack Compute.

quantum API
API used to access quantum, provides and extensible architecture to allow custom plugin creation.

quantum manager
Allows nova and quantum integration thus allowing quantum to perform network management for nova VMs.

quantum plugin
Interface within quantum that allows organizations to create custom plugins for advanced features such as QoS, ACLs, or IDS.

quarantine
If swift finds objects, containers, or accounts that are corrupt they are placed in this state, are not replicated, cannot be read by clients, and a correct copy is re-replicated.

Quick EMUlator (QEMU)
One of the hypervisors supported by OpenStack, generally used for development purposes.

quota
In nova, the ability to set resource limits on a per-project basis.

RAM filter
The nova setting that allows or disallows RAM overcommitment.

Recon
A component of swift used to collect metrics.

RAM overcommit
The ability to start new VM instances based on the actual memory usage of a host, as opposed to basing the decision on the amount of RAM each running instance thinks it has available. Also known as memory overcommit.

record ID
A number within a database that is incremented each time a change is made. Used by swift when replicating.

registry server
A glance service that provides VM image metadata information to clients.

rate limit
Configurable option within swift to limit database writes on a per-account and/or per-container basis.

replica
Provides data redundancy and fault tolerance by creating copies of swift objects, accounts, and containers so they are not lost when the underlying storage fails.

rebalance
The process of distributing swift partitions across all drives in the ring, used during initial ring creation and after ring reconfiguration.

replica count
The number of replicas of the data in a swift ring.

replication
: The process of copying data to a separate physical device for fault tolerance and performance.

replicator
: The swift back-end process that creates and manages object replicas.

request ID
: Unique ID assigned to each request sent to nova.

ring
: An entity that maps swift data to partitions. A separate ring exists for each service, such as account, object, and container.

ring builder
: Builds and manages rings within swift, assigns partitions to devices, and pushes the configuration to other storage nodes.

S3
: Object storage service by Amazon, similar in function to swift, can act as a back-end store for glance VM images.

scheduler manager
: A nova component that determines where VM instances should start. Uses modular design to support a variety of scheduler types.

scoped token
: A keystone API access token that is associated with a specific tenant.

secret key
: String of text only known by the user, used along with an access key to make requests to the nova API.

security group
: A set of network traffic filtering rules that are applied to a nova instance.

segmented object
: A swift large object that has been broken up into pieces, the re-assembled object is called a concatenated object.

role ID
: Alpha-numeric ID assigned to each keystone role.

rootwrap
: A feature of nova that allows the unprivileged "nova" user to run a specified list of commands as the Linux root user.

RPC driver
: Modular system that allows the nova underlying message queue software to be changed. For example, from RabbitMQ to ZeroMQ or Qpid.

server image
: Alternative term for a VM image.

server UUID
: Unique ID assigned to each nova VM instance.

service catalog
: Alternative term for the keystone catalog.

service ID
: Unique ID assigned to each service that is available in the keystone catalog.

service registration
: A keystone feature that allows services such as nova to automatically register with the catalog.

service tenant
: Special keystone tenant that contains all services that are listed in the catalog.

service token
: An administrator defined token used by nova to communicate securely with keystone.

session back-end
: The method of storage used by horizon to track client sessions such as local memory, cookies, a database, or memcached.

session persistence
: A feature of the load balancing service. It attempts to force subsequent connections to a service to be redirected to the same node as long as it is online.

session storage
: A horizon component that stores and tracks client session information. Implemented through the Django sessions framework.

shared storage
: Block storage that is simultaneously accessible by multiple clients. For example, NFS.

SmokeStack
: Runs automated tests against the core OpenStack API, written in Rails.

snapshot
: A point-in-time copy of an OpenStack storage volume or image. Use storage volume snapshots to back up volumes. Use image snapshots to back up data, or as "gold" images for additional servers.

spread-first scheduler
: The nova VM scheduling algorithm that attempts to start new VM on the host with the least amount of load.

SQLAlchemy
: An open source SQL toolkit for Python, used in OpenStack.

SQLite
: A lightweight SQL database, used as the default persistent storage method in many OpenStack services.

StackTach
: Community project that captures nova AMQP communications, useful for debugging.

static IP address
: Alternative term for a fixed IP address.

StaticWeb
: WSGI middleware component of swift that serves container data as a static web page.

storage back-end
: The method that a service uses for persistent storage such as iSCSI, NFS, or local disk.

storage node
: A swift node that provides container services, account services, and object services, controls the account databases, container databases, and object storage.

storage manager
: Component of XenAPI that provides a pluggable interface to support a wide variety of persistent storage back-ends.

storage manager back-end
: A persistent storage method supported by XenAPI such as iSCSI or NFS.

storage services
: Collective name for the swift object services, container services, and account services.

swift
: An OpenStack core project that provides object storage services.

swift All in One (SAIO)
: Creates a full swift development environment within a single VM.

swift middleware
: Collective term for components within swift that allows for additional functionality.

swift proxy server
: Acts as the gatekeeper to swift and is responsible for authenticating the user.

swift storage node
: A node that runs swift account, container, and object services.

sync point
Point in time since the last container and accounts database sync among nodes within swift.

TempAuth
An authentication facility within swift that allows swift itself to perform authentication and authorization, frequently used in testing and development.

Tempest
Automated software test suite designed to run against the trunk of the OpenStack core project.

TempURL
A swift middleware component that allows a user to create URLs for temporary object access.

tenant
A group of users, used to isolate access to nova resources. An alternative term for a nova project.

tenant endpoint
A keystone API endpoint that is associated with one or more tenants.

unscoped token
Alternative term for a keystone default token.

updater
Collective term for a group of swift components that process queued and failed updates for containers and objects.

user
In keystone each user is associated with one or more tenants, and in nova they can be associated with roles, projects, or both.

VIF UUID
Unique ID assigned to each quantum VIF.

Virtual Central Processing Unit (vCPU)
Allows physical CPUs to be sub-divided and those divisions are then used by instances. Also known as virtual cores.

tenant ID
Unique ID assigned to each tenant within keystone, the nova project IDs map to the keystone tenant IDs.

token
An alpha-numeric string of text used to access OpenStack APIs and resources.

tombstone
Used to mark swift objects that have been deleted, ensures the object is not updated on another node after it has been deleted.

transaction ID
Unique ID assigned to each swift request, used for debugging and tracing.

user data
A blob of data that can be specified by the user when launching an instance. This data can be accessed by the instance through the metadata service or config drive. Commonly used for passing a shell script that is executed by the instance on boot.

Virtual Machine (VM)
An operating system instance that runs on top of a hypervisor. Multiple VMs can run at the same time on the same physical host.

virtual network
An L2 network segment within quantum.

Virtual Network InterFace (VIF)
An interface that is plugged into a port in a quantum network. Typically a virtual network interface belonging to a VM.

virtual port
Attachment point where a virtual interface connects to a virtual network.

virtual private network (VPN)
Provided by nova in the form of cloudpipes, specialized instances that are used to create VPNs on a per-project basis.

virtual server
Alternative term for a VM or guest.

virtual switch (vSwitch)
Software that runs on a host or node and provides the features and functions of a hardware based network switch.

virtual VLAN
Alternative term for a virtual network.

VLAN manager
A nova networking manager that divides subnet and tenants into different VLANs allowing for Layer 2 segregation. Provides a DHCP server for each VLAN to assign IP addresses for instances.

VLAN network
The Network Controller provides virtual networks to enable compute servers to interact with each other and with the public network. All machines must have a public and private network interface. A VLAN network is a private network interface, which is controlled by the vlan_interface option with VLAN managers.

VM image
Alternative term for an image.

VNC proxy
A nova component that provides users access to the consoles of their VM instances through VNC or VMRC.

volume
Disk-based data storage generally represented as an iSCSI target with a file system that supports extended attributes, can be persistent or ephemeral. Commonly used as a synonym for block device.

Volume API
An API on a separate endpoint for attaching, detaching, and creating block storage for compute VMs.

volume controller
A nova component that oversees and coordinates storage volume actions.

volume driver
Alternative term for a volume plugin.

volume ID
Unique ID applied to each storage volume under the nova control.

volume manager
A nova component that creates, attaches, and detaches persistent storage volumes.

volume node
A nova node that runs the cinder-volume daemon.

volume plugin
A plugin for the nova volume manager. Provides support for a new and specialized types of back-end storage.

Volume Service API
Alternative term for the Block Storage API.

volume worker
The nova component that interacts with back-end storage to manage the creation and deletion of volumes and the creation of compute volumes, provided by the nova-volume daemon.

weight
: Used by swift storage devices to determine which storage devices are suitable for the job. Devices are weighted by size.

weighted cost
: The sum of each cost used when deciding where to start a new VM instance in nova.

weighing
: A nova process that determines the suitability of the VM instances for a job for a particular host. For example, not enough RAM on the host, too many CPUs on the host, and so on.

worker
: A daemon that carries out tasks. For example, the nova-volume worker attaches storage to an VM instance. Workers listen to a queue and take action when new messages arrive.

Zuul
: Tool used in OpenStack development to ensure correctly ordered testing of changes in parallel.