Chapter 1 – OpenStack and Cloud Architecture Patterns

"We start, as always, at the beginning" – James Lipton

"I don't need a hard disk in my computer if I can get to the server faster…carrying around these non-connected computers is byzantine by comparison." – Steve Jobs

In this chapter, we will:

- Discuss the content of the book
- Review cloud architecture patterns
- Discuss REST and APIs
- Discuss the OpenStack foundation and community
- Review the OpenStack releases and projects

Welcome to Mastering OpenStack! To master any product, we need to understand how it came to be. Our journey begins with a look back at the inception of the OpenStack project and how we have come to arrive at the release of Grizzly, the current stable release of the product which was delivered in April 2013.

For those who are new to OpenStack we will be covering the concepts at a variety of detail levels so it will be possible to use Mastering OpenStack as a resource guide for new and existing OpenStack administrators. We highly recommend that you also read the OpenStack Cloud Computing Cookbook which is also available from Packt Publishing as a companion book.

As we work through the book, we will explore OpenStack with a detailed focus on each of the core components to ensure that the reader comes away with a full understanding of the OpenStack framework.

Technical examples are provided, along with topologies for the examples to provide context for our scenarios. We will be using multiple topologies to be able to evaluate different deployments and different business and technical requirements.

I encourage the “play-along at home” method wherever possible. The samples provide here in the guide to allow you to work through command line samples and scripts to grow your comfort with OpenStack.

Before we begin, here is a brief synopsis of the chapters:

Chapter 1: OpenStack and Cloud Architecture Patterns

We discuss the high level view of the OpenStack projects and the origin of OpenStack as well as a review of cloud architecture patterns. This is the basic foundation before we get in to the detailed OpenStack component architecture.

Chapter 2: Planning your OpenStack Deployment

We discuss the prerequisites and requirements for our deployment including the lab requirements and sizing for your production deployments. We will cover automation tools as well as standalone deployment methods to get the environment configured. Single node (All-in-One) and multi-node will be discussed plus we touch upon vendor implementations such as DevStack and the Rackspace Private Cloud.

Chapter 3: OpenStack Identity – Keystone

A deep view of the Keystone Identity service, including management and usage patterns for the Keystone API. We will explore enabling LDAP integration, the Keystone CLI commands and SSL integration.

Chapter 4: OpenStack Storage – Swift and Cinder

Storage services inside the OpenStack environment will be an important portion of our guide. A review of the high level concepts of block and object storage will start us off and be followed by a deeper dive into use cases for each.

Performance and tuning of the Swift environment as well as the use of swift-bench for benchmarking the implementation will be expanded upon. The Cinder block storage architecture and management will also be discussed with comparisons provided to similar block storage environments in alternate cloud and server environments.

Chapter 5: OpenStack Image – Glance

Using the Glance services, we explain how images are stored, managed and deployed and how the Glance API is used. There will be discussion on the commonly used images and their use cases which will touch briefly on creating images (greater detail to come in the later chapter).

Chapter 6: OpenStack Compute – Nova

The core of the OpenStack environment, our chapter will delve into the Nova architecture including the core networking deployment architecture, security and the dashboard features of Horizon. Understanding the Nova environment will be emphasized with a breakdown of the hypervisor support and features available in each particular hypervisor.

Chapter 7: Creating Custom Images

Explore the creation process for custom images including Windows based images and how to use the tools to deploy your new images into your OpenStack environment. Tools and techniques are brought in for using existing images from other hypervisors and optimization tips for your custom images.

Chapter 8: Advanced Networking – Neutron Configuration

Deeper exploration of the Neutron networking features including Open vSwitch integration, plugins for third party providers and implementation examples for more complex network designs. Security and network boundaries will be clearly described with use case examples for each architecture pattern.

Chapter 9: High Availability OpenStack Deployments

Architectural patterns for building high availability systems using OpenStack. This chapter will provide detailed design specifications and discuss the challenges of providing high

availability services. There are clear deficiencies in many implementations which will be highlighted here to provide real examples of where dependencies are created and failures can occur.

Chapter 10: Troubleshooting OpenStack

Troubleshooting tools and techniques including log management, process management and command line tips for finding and dealing with issues in your OpenStack deployment. An often overlooked subject, we will be looking at the contents of logs, how to collect meaningful information from them and how to effectively search for solutions.

Chapter 11: What's next with OpenStack?

An overview of the upcoming features and changes slated for the Havana release and some of the important next steps for the OpenStack community.

Appendix A: Resources

A list of online resources and some key OpenStack authors and bloggers are provided which will help you with your OpenStack learning and administration experience.

Appendix B: Glossary of Terms

Terms and definitions that are used in the book and in the OpenStack and cloud ecosystem.

Before we dive into OpenStack itself, let's review some core cloud concepts so that we can understand how it is that OpenStack, its features, and its individual components will be deployed and how they came into being. For many of you, this may be a review of some well-known concepts, but our goal is to make sure that everyone is brought up to the same point and that our terminology is well understood.

Cloud Architecture Patterns

While this section may be review for many, it is a worthwhile step to level everyone up on definitions and requirements when we refer to different cloud architectures. Having an understanding of common cloud concepts is the basis for how we evaluate the OpenStack architecture.

The simple start to it all is knowing the basic tenets of a cloud environment. The NIST (National Institute of Standards and Technology) has defined five characteristics of a cloud computing environment:

- On-Demand self-Service
- Broad network access
- Resource pooling
- Rapid elasticity
- Measured service

These key characteristics will all be considered when we discuss cloud computing infrastructure, and your OpenStack environment will be fully capable of delivering all of the above requirements. Rapid elasticity is subject to hardware constraints in a public cloud environment, but we will discuss that further later.

Now we will take a look at the three categories of cloud environments to understand the differences in the design and features of each.

Public Cloud

A public cloud infrastructure is provisioned by a service provider with services available using a pay-per-use model. The public cloud infrastructure model is fully managed by the vendor and the consumer accesses all resources over public networks.

Amazon Web Services is often the first that comes to mind when discussing the public cloud environments because there were one of the first large-scale public cloud providers. Rackspace is now a well-known public cloud provider which uses OpenStack for their underlying infrastructure.

The assumption with public cloud infrastructure is that it can scale to meet any resource requirement. We understand that there are obvious limits to how quickly they can

respond to growth, but for all intents and purposes we regard public cloud as being without growth limitations.

Many ASPs (Application Service Providers) use the public cloud infrastructure because of their rapidly growing demand and potential to also reduce services when a lull in traffic occurs, such as seasonal offerings like tax refund services.

Private Cloud

Private cloud infrastructure is managed and hosted on-premises. While a vendor may be involved in the management of the environment, the key requirement is that this is entirely on-premises at the customer site. Common reasons for choosing a private cloud model are privacy and security.

Some organizations have strict regulations on where data can be stored and who can have access to the environment. Private cloud environments are typically more expensive to deploy because of the design and infrastructure requirements.

A popular challenge in the on-premises environment is differentiating between a virtualized datacenter and a cloud datacenter. They are distinctively different. Recall that the requirement for our cloud is that it is a self-service model with elastic resources available on-demand. The immediate requirement is that the applications and services can be requested through a web resource or dashboard, so if you have any interaction in your request process that is phone or email based, I've got bad news for you; you aren't running a private cloud.

An often argued point about private cloud deployments, is the ability if the IT organization to respond to rapid growth. While I agree that a sudden, unplanned, massive increase in resource requests may overwhelm a cloud datacenter, the assumption is that the IT organization works to monitor and manage resource consumption and is well prepared for sudden increases in resourcing needs.

Hybrid Cloud

A fast growing cloud deployment model, the hybrid cloud is a mixture of private cloud and public cloud infrastructure which allows for the security and control of on-premises, while

also connecting to a public cloud for portions of the infrastructure to leverage the elastic capability for applications which can make use of it.

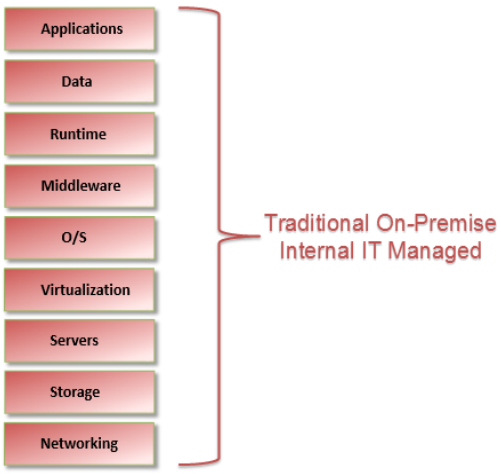
As with private cloud deployments, there are the costly requirements of hosting a full infrastructure on-premises. The advantage of mixing the public and private deployment models is that we can fully realize both the security of the private model with the elastic capability of the public infrastructure. This is often a stepping stone for many organizations to move towards a primarily public cloud model.

An even more exciting feature of many cloud infrastructure products is the ability to leverage other vendor products as a part of the build. For example, we can deploy our OpenStack cloud with Amazon S3 storage so that we do not need to create our own storage infrastructure which lets us maintain the control and management in our OpenStack environment.

Cloud Service Models

Beyond the three categories of cloud architecture, we also should know about the *aaS, or as-a-Service categories to be sure we are fully onboard with those too.

Using the service stack shown below we see the different services that are provided from the network up to the application layer. This shows the on-premises deployment which is fully managed by the internal IT departments:

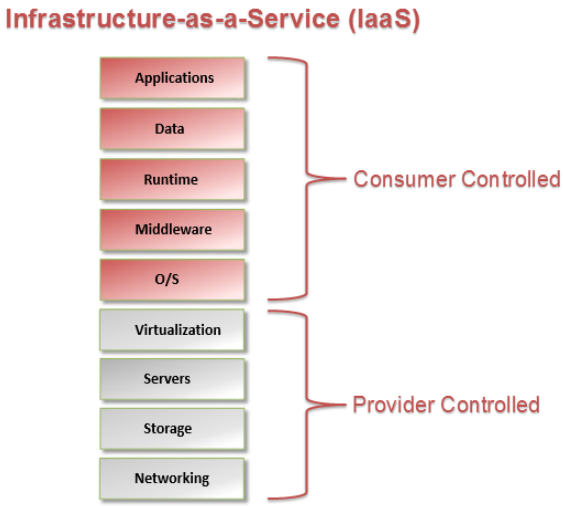


INSERT 9376OS_01_01.PNG

As you can see, the entire service stack is managed internally. This is what we know as the traditional model of IT service management, but for what we are discussing of course, this is a private cloud deployment model.

IaaS - Infrastructure-as-a-Service

With IaaS platforms, the networking, storage, server and virtualization layer are handled by the provider. This is the model delivered by Amazon EC2 and Google Compute Engine as an example. This is also what the consumer of OpenStack services will receive as provided by vendors such as Rackspace.



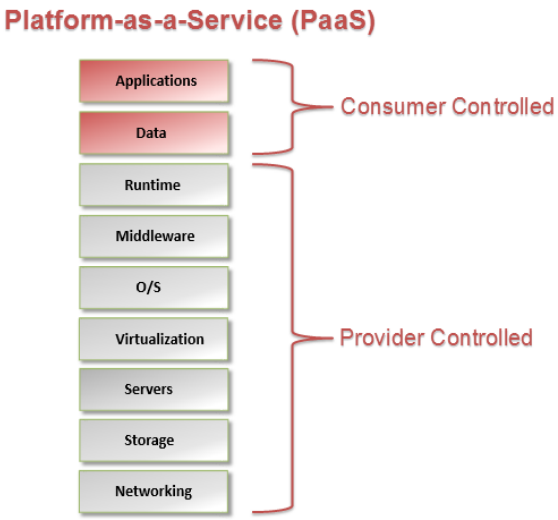
INSERT 9376OS_01_02.PNG

To the consumer, there is no need to be aware of the bottom layers of the stack because they have no control at those layers and they simply consume resources from the Operating System upwards.

As discussed earlier, all services are provided through self-service portals using a service catalog. Different flavors of O/S are available as well as some pre-built application images which reduce the administration even more for the consumer.

PaaS - Platform-as-a-Service

Looking further up the stack, PaaS providers such as Cloud Foundry and Heroku deliver application hosting infrastructure based on the specific development platform. The O/S, middleware and runtime are now fixed and the only consumer managed portion is the application code.



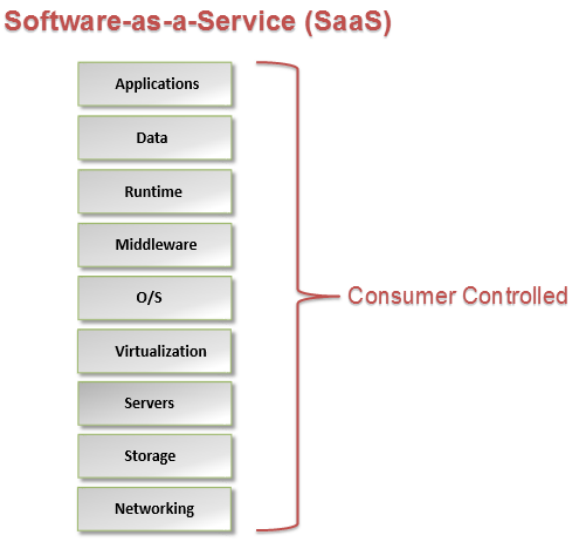
INSERT 9376OS_01_03.PNG

Application and data instances are provisioned and controlled through APIs. Some services even provide locally hosted infrastructure to allow developers to run instances which can be deployed into the PaaS environment in the cloud environment.

This is a fast growing sector among development communities as it provides the capability for rapid application deployment and embraces the DevOps practices which have become a building block of successful IT organizations worldwide.

SaaS – Software-as-a-Service

We know many popular names in this space of course. As the most vendor-managed cloud platform, the SaaS applications include Salesforce.com, Google Apps, and Office 365 among many others.



INSERT 9376OS_01_04.PNG

Beware the wolf in sheep's clothing though. Not all internet based applications are Software-as-a-Service deployments. There are still many application vendors who offer hosted solutions, but they are not, by definition, cloud applications.

Much like the requirements to define a cloud computing environment, the SaaS requirements tie in strongly to service level agreements. Application hosting should leverage the cloud fully to provide redundancy and resiliency for the customer. Simply running a web application on a virtual private server instance does not meet the SaaS requirements.

DBaaS – Database-as-a-Service

Although Database-as-a-Service is really a subset of the PaaS, it is worthwhile mentioning it on its own. There are providers of DBaaS for most significant on-premises database vendors such as Microsoft SQL, MySQL and many NoSQL variations.

There is a blueprint for an OpenStack DBaaS, although the project has not been started at the time of this writing. I predict that we will be seeing a lot of growth in this

environment over the coming months. Creating and deploying resilient database services for consumption in the cloud is an exciting concept and also requires some significant design considerations.

Don't confuse running a MySQL server in a cloud deployment as a single server as a DBaaS. The requirement for a DBaaS deployment is not just the storage mechanism, but also the entire transaction handling and message queuing infrastructure.

DBaaS can be a traditional RDBMS (e.g. MSSQL, MySQL) or a NoSQL environment (e.g. MongoDB, CouchDB) also. The variety of DBMS flavors is increasing month by month and many new vendors are providing cloud DBaaS products which is a positive step in providing options for customers.

NaaS – Network-as-a-Service

Truthfully, this could be an entire chapter unto itself. The SDN (Software Defined Networking) and NV (Network Virtualization) world is quickly filling with supporting vendors and products.

OpenStack makes use of its own core networking products, as well as providing plugins for a number of significant network vendors such as Nicira and the widely popular Open vSwitch environment.

Network-as-a-Service is typically seen as VPN connectivity, bandwidth-on-demand and services such as load balancing, which itself even gets classified on its own as LBaaS (Load Balancing-as-a-Service) in many cases.

This will be a significant discussion in the book as we have many design and deployment considerations for creating a NaaS environment. We will be touching on traditional network designs in a virtual network, and then moving to more advanced topics such as stretched clustering, multi-tenant firewall designs, tunneling protocols and more.

We have great flexibility with the services, vendors and methods to deploy and manage SDN infrastructure to support the OpenStack ecosystem. This is a significant part of the book as it is one of the most popular, and also one of the most polarizing topics when discussing OpenStack deployments.

Network architecture is a particularly exciting challenge because network topologies can be “right” or “wrong” in many ways depending on circumstances and requirements. NaaS is a platform design to eliminate the need for the consumer of the service to do complex design in their own infrastructure. As we explore the features of OpenStack networking, we will build the foundation for creating robust network scenarios to meet those needs.

Living in a Software Defined World

The hot topic over the last couple of years is the buzzword-worthy phrase Software Defined. We have to be careful when we use Software Defined to refer to infrastructure components because there are, or at least there should be, rules on what is qualified to carry that title.

Software Defined implies the system is a software abstraction of what was previously a hardware based system. It is not that we are removing the basic physical infrastructure that we have had for decades, but now we are creating virtual data centers where the roles and features are now encapsulated within a virtual system.

Software Defined Networking (SDN) and the Software Defined Data Center (SDDC) are definitely the hottest topics in the last couple of years. As more vendors have entered the market with cloud based services, there are more data center environments which are operating at the software layers.

Much like how server virtualization separated the underlying hardware from the guest operating system, now we are separating the network and storage layers from the guest as well. By doing this we gain flexibility for what we use at the hardware layer because it is now less important.

Vendors are touting the use of commodity hardware as the next way to bring cloud infrastructure to organizations with on-premises and public hosted environments where we would normally have relied on expensive, proprietary systems that had genuine limitations with scalability without high costs and complex designs.

Now we can use loosely coupled systems which abstract the hardware layer we can use new tools and techniques to build more robust, flexible and scalable systems that no longer rely on vendor locked-in hardware solutions to extend the data center and our virtualization strategy.

The Importance of the API

The way that we link to these loosely coupled systems is with addressable software layers. And so we begin with the API (Application Programming Interface).

APIs are common in many infrastructure systems today. The implementation of an API allows vendors, partners and consumers of a service to leverage the native capabilities of a system without having to program directly in the same language or have a tightly coupled, rigid dependence on the original system.

Software systems with an exposed, well documented API represent the future of well-crafted systems infrastructure. As we will see throughout this guide, the OpenStack ecosystem thrives on the fact that it is API based, and even more exciting is that it is entirely open. Note that open infrastructure does not necessarily guarantee flexibility and future compatibility, however the OpenStack environment is being developed with this in mind and throughout the evolution up to this point the contributors have maintained consistency with the APIs.

By providing open APIs for the OpenStack architecture, developers can easily upgrade and strengthen the core of each portion of the OpenStack environment because each of the systems will continue to consume the resources using standard API instructions.

Each of the OpenStack components comes with a fully documented API. Any interaction within the systems is API based. In doing so, we allow for the opportunity to do granular upgrades within the environment without affecting operational stability. Certain API commands will be deprecated over time and functionality will be migrated as new features and functions are made available.

All communication done between the components is done via the API. This is a challenging concept for some to understand, but as we look at our cloud systems, we will see time and time again that the key factor to maintain interoperability is the use of the API to interact with the system.

Because the API is used, we also gain flexibility to create our own systems which can simply leverage the API to perform tasks. You can extend a web-based request system to request and manage resources with your OpenStack deployment, or create a shim to

capture chargeback and monitoring information to pull the content into an existing service management tool that you have in your environment already.

REST is Best

For those not yet familiar, REST (REpresentational State Transfer) is derived from HTTP standards which ensure that resources are fully addressable and that command structures are common across all resources.

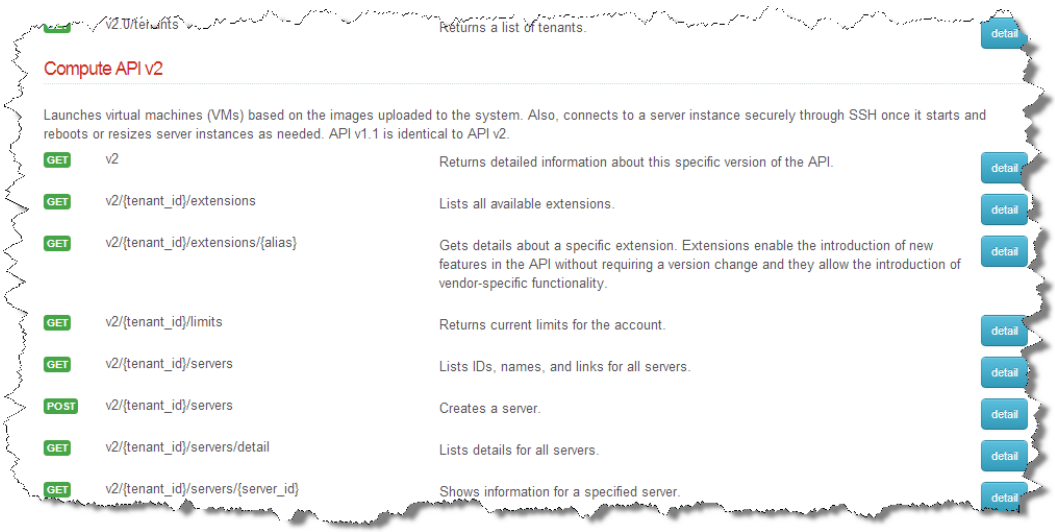
Roy Fielding, co-founder of the Apache HTTP Server project, was also one of the principal authors of the HTTP specification. He produced a widely read and referenced dissertation on REST (http://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm) which lays out the clear rules and definitions of what makes a RESTful interface and the reason for utilizing the REST methods for your API.

Detailed API reference documents are available for the OpenStack ecosystem on the OpenStack documentation site at http://docs.openstack.org/api/. As we proceed through the book there will be API discussions for each of the OpenStack components. These are integral to understanding how to extend your OpenStack knowledge.

Because we are using RESTful addressing, we can issue commands using a browser, or a command line HTTP request. This is an example of viewing a list of servers in your tenant cloud using the cURL utility:

```
Curl –v –H “X-Auth-Token:yourtokenhere”
http://yourcloudhost:8774/v2/tenantnn/servers
```

The URL structure is consistent as we will see during our exploration of the individual OpenStack components. Full API documentation is published at http://api.openstack.org/api-ref.html:



INSERT 9376OS_01_17.PNG

Each of the chapters on the OpenStack projects will have API usage reference and we will be using the RESTful URLs for managing our resources. Because of the RESTful resource management, you can quickly learn how to interact with your OpenStack systems through HTTP which is a powerful and flexible feature.

The Origin of OpenStack

How OpenStack Started

Everyone loves a Cinderella story, and this may be just that. What began as a joint venture with Rackspace Hosting and NASA (National Aeronautics and Space Administration) in July 2010, has grown into a widely adopted open source cloud hosting platform with over 150 vendors contributing to the project.

The decision to release the product through the open source community has been a boon for the cloud community and vendors alike. Now, with important contributors involved in

the development of OpenStack, we are watching the steady growth and development of a robust, scalable, enterprise cloud system that will continue to gain adoption and support.

Now in its third year of production implementation, the OpenStack framework (often called the OpenStack cloud operating system) is becoming widely utilized by organizations and service providers, and is supported by numerous consultants and vendors in both public and private implementations.

Since its inception, multiple OS vendors have partnered to be able to provide OpenStack implementations using open source, and commercially supported operating systems. As mentioned earlier in the chapter, we will use Ubuntu for most of our examples, but you may have more comfort with using Red Hat or SUSE Linux. There are also distributions for Debian and Fedora, but to maintain consistency we will concentrate on the OpenStack components themselves rather than the underlying operating systems.

OpenStack Contributors

As was mentioned earlier, there are over 150 vendors who contribute to the project at the time of this writing. These include names such as IBM, HP, Cisco, VMware, Intel and AMD just to name a few.

The developer community is a strong group of development experts, with growth in the number code contributors with each release. This development model is the same strong, community model which has driven the Linux ecosystem since its inception.

For anyone who wants to contribute to the project, you can go to the Launchpad site (https://launchpad.net/openstack) to join the lists and view the project status and bug tracking information which is maintained by the development team and the OpenStack community.

OpenStack Foundation

The OpenStack Foundation is charged with maintaining the technical, strategic and financial leadership for the OpenStack projects.

A technical board consisting of 13 members provide guidance for the project stewards of each of the OpenStack components. As an elected board, the team gives the leadership needed to align the over 500 developers of the platform.

With the members of the technical committee and the board of directors coming from various vendors and backgrounds, the community gets a broad base of knowledge and skill brought to the project and this has proven to be a winning formula.

The user community also drives the growth and feature management with over 5600 individual members and 850 organizations involved. There are few platforms which are being developed with this scale of involvement.

OpenStack Hosting Vendors

OpenStack was dubbed The Open Source Cloud Operating System, and is not limited to on-premises or purely public cloud deployments. The environment is used for private, public and hybrid cloud deployments around the world, spanning every industry sector.

For public hosting of your OpenStack infrastructure, there are many vendors such as Rackspace, Piston and HP. The list of vendors is growing to include those who use OpenStack as their underlying technology to provide IaaS, PaaS and SaaS deployments in every shape and size you can imagine.

Hybrid deployments aren't widely documented at the time of this writing, but they certainly do exist and we will discuss some of the requirements and techniques to deploy hybrid installations using the OpenStack framework. With the rise in popularity of OpenStack, there are more and more participating vendors who are joining the support ecosystem which is a sign of great things to come.

Release History

Already at their 7th full release, with Havana scheduled for delivery in late 2013, the product has been furiously deploying product releases since the launch of Austin in 2010.

Series	Status	Releases	Date
Havana	Under development	Due	Oct 17, 2013
Grizzly	Current stable release, security-supported	2013.1	Apr 4, 2013
		2013.1.1	May 9, 2013
Folsom	Security-supported	2012.2	Sep 27, 2012
		2012.2.1	Nov 29, 2012
		2012.2.2	Dec 13, 2012
		2012.2.3	Jan 31, 2013
		2012.2.4	Apr 11, 2013
Essex	EOL	2012.1	Apr 5, 2012
		2012.1.1	Jun 22, 2012
		2012.1.2	Aug 10, 2012
		2012.1.3	Oct 12, 2012
Diablo	EOL	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

INSERT 9376OS_01_05.PNG

As with many agile application projects, the features that are included are decided upon throughout the development cycle and then finalized for full releases in iterative cycles. What began as two components (Nova and Swift) has now been broken out into 7 distinct projects with many others in what is referred to as incubation.

Incubated projects may or may not become adopted in each release, and may morph into other features or projects as development continues. Our focus is the code, and projects that are included in the April 2013 release known as Grizzly but we will look at some of the other key projects that are on their way to being part of the core group in Havana.

What's new in the Grizzly Release?

Looking back to the Folsom release, there were significant changes including the addition of Neutron (formerly Quantum) and Cinder as individual projects. Now we have seen another massive jump in functionality and features. The seventh release of the OpenStack system has 230 additional features that have come since the Folsom release in 2012.

We won't be listing out the full set of features that were new, however you can go to the OpenStack Launchpad site (https://Launchpad.net/OpenStack) for detail on code commits and the entire bug and feature tracking.

Feature Highlights in Grizzly

Among the 230 new features added, we have highlighted some exciting features included in Grizzly which are key in making the OpenStack ecosystem more robust and scalable.

- Keystone API V3
- Token plus string authentication support
- Fibre Channel and FCoE support for block storage
- OpenStack Networking support added for systems including Hyper-V and Brocade
- Increased networking support for Open vSwitch, Cisco UCS, Cisco Nexus and Nicira among others
- Load Balancing-as-a-Services (LBaaS)
- Better Glance image sharing and discoverability
- More robust hypervisor support for VMware ESX, KVM, Xen and Microsoft Hyper-V
- Object storage performance advancements
- Bulk object storage operations support added to enable the use of large storage clusters
- Direct browser to back-end object storage access added
- Significant new features in the OpenStack networking framework which lay the groundwork for more vendor support

So from this brief view of new features, we can see that the feature set is increasing in a number of ways. What is very encouraging is the inclusion of many enterprise network, compute and storage vendors in the contributors. The adoption by these vendors ensures that organizations will be able to leverage their existing hardware while enjoying the full support of the vendor which can be a limiting factor for adoption by many.

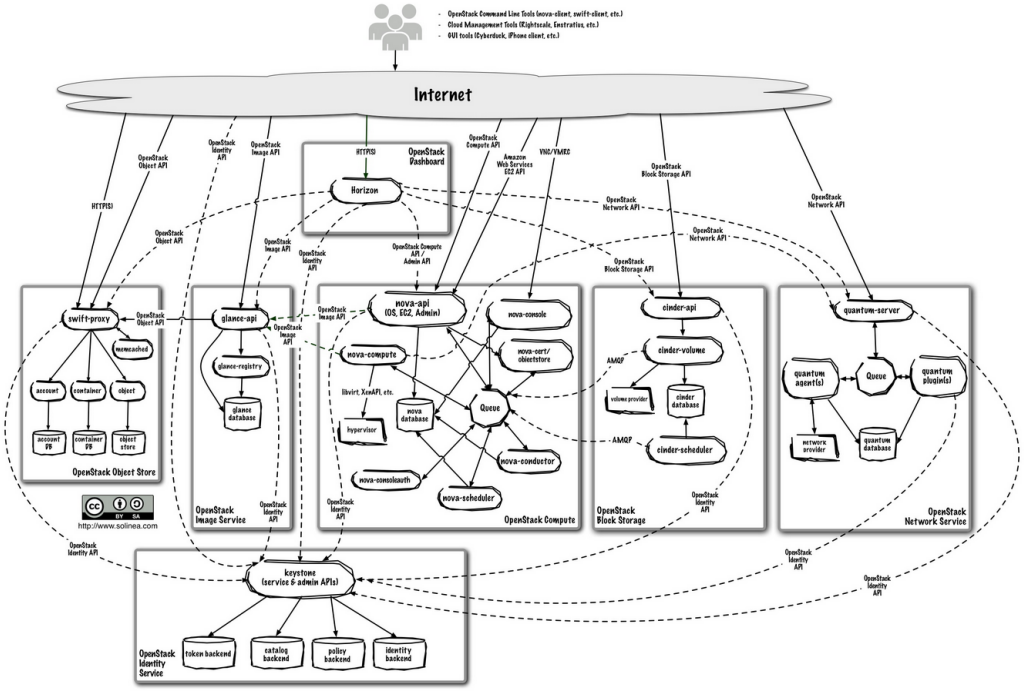
Contrary to the HCL (Hardware Compatibility List) of other hypervisor and server OS vendors, the OpenStack requirements are minimal. What we see with the increase in

vendor-specific driver and module development is the ability to fully leverage the underlying features of the hardware capabilities of these products.

OpenStack Architecture

To understand the overall OpenStack architecture, we will look at the different components involved which make up the overall ecosystem. These are referred to as “projects”.

In the diagram we see the components of OpenStack and how they interact with each other. As you can see, this is an incredibly detailed diagram, and also an eye test in print unfortunately. Each component will be displayed in the project detail section below.



INSERT 9376OS_01_06.PNG

Diagrams are provided by the OpenStack.org site and can be viewed online at http://docs.openstack.org/grizzly/openstack-compute/admin/content/ch_getting-started-with-openstack.html

The OpenStack Projects

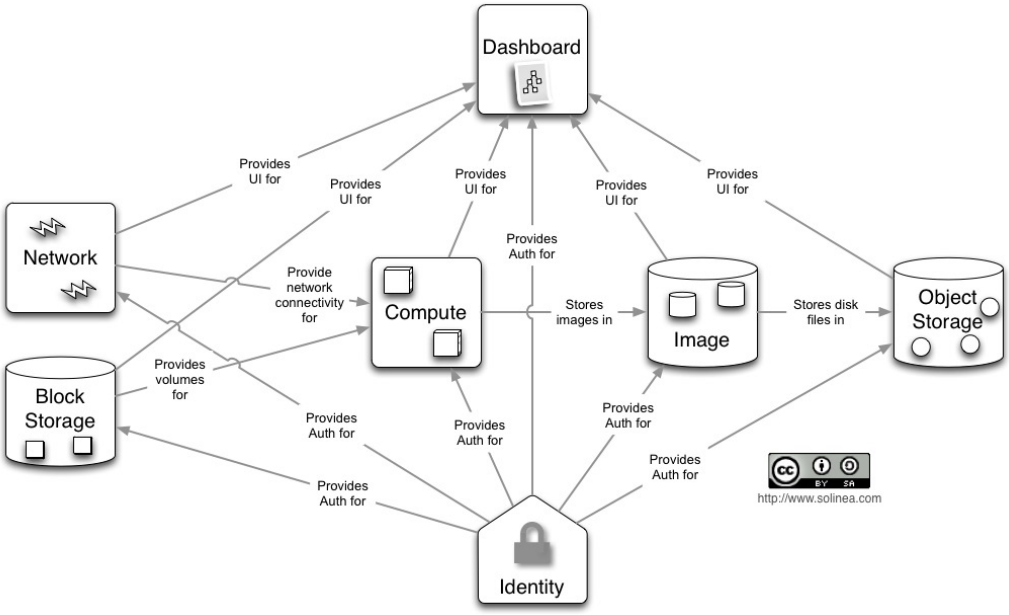
Now that we've covered the fundamentals and history of the OpenStack ecosystem, it is time to start diving into the core components. Each of the projects we discuss are fully deployed projects which through the evolution have been spun off from what was once part of the core project.

There are standard projects which make of the core features of OpenStack, as well as other projects which are called "incubation" projects. Incubation projects are components which are in development but are not supported. Effectively the incubation projects are pre-1.0 version features which will be developed, tested and once they reach an acceptable level of maturity, will become part of the Common Projects.

At its inception, the OpenStack consisted of two core projects which were the compute environment (Nova) and the object storage environment (Swift). The product was an evolution of the Rackspace Cloud Files service, thus the inclusion of these two key features.

With the Grizzly release, there are now 7 projects actively being managed, with many others in incubation. For our purposes we want to concentrate on the core projects, although we will touch on a few of the additional features that are in the incubated projects as we progress through the book.

Here is a simplified view of the core components and their relationships:

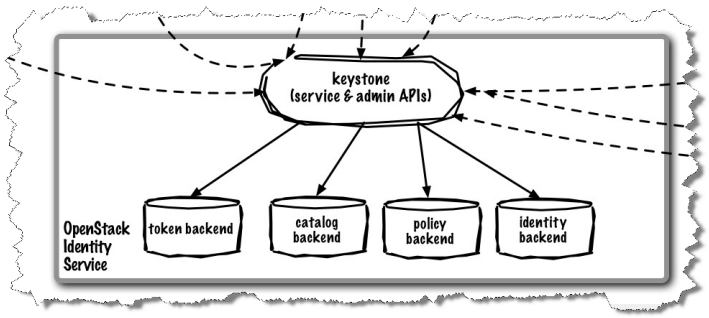


INSERT 9376OS_01_16.PNG

Now let's take a look at each of the components to understand what it is that they do.

OpenStack Identity – Keystone

The keys to the kingdom lie within the Keystone application. This is the authentication and authorization engine for OpenStack. Authentication with Keystone is done using a username and password combination, which then utilizes a token, which is passed between the compute and storage environments to initiate any actions.



INSERT 9376OS_01_07.PNG

Using Keystone, we will manage users, accounts and roles for our OpenStack environment. For this reason, we start with the Keystone services in order to create the security infrastructure before we build out the rest of the infrastructure components.

Much like the keystone in a bridge, the identity component is the integral connector between all of the OpenStack systems. Every transaction initiated by the user is validated with their token against the target component to confirm authentication, and most importantly, authorization to be able to complete the requested task.

We will explore the various configuration options such as the database configuration, SSL certificate installation and management, as well as LDAP configuration for existing backend authentication environments.

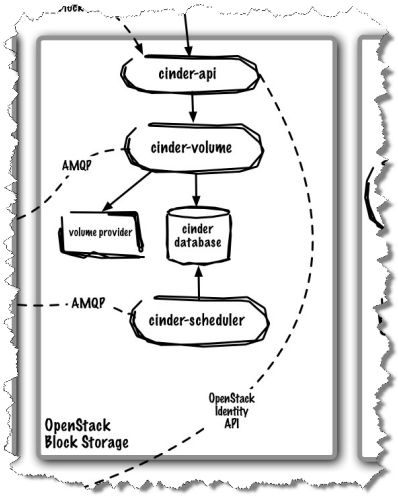
OpenStack Block Storage service – Cinder

Block storage in the OpenStack environment is provided by the appropriately named Cinder. Block devices provide persistent storage to compute resources which was done using iSCSI presentation prior to the Grizzly release, but now supports SAN storage using FC and FCoE.

Many new storage vendors, including Ceph, HP, Coraid, EMC, NetApp, IBM, and GlusterFS among others are supported as backend providers as a result of community developed drivers.

The block storage environment also has snapshot functionality to support the backing up of data to use as restore points, or to create new volumes from the snapshot images.

This function provides a redundancy and protection capability for the consumer and presents possibilities for BCP/DR protection products to integrate with OpenStack using the native API support.



INSERT 9376OS_01_08.PNG

Cinder volumes are persistent storage and are retained until deleted by the user or administrator. This differs from the ephemeral storage which is assigned to a VM while it is running. Ephemeral storage is used for scratch space and swap space and is removed when a VM is terminated.

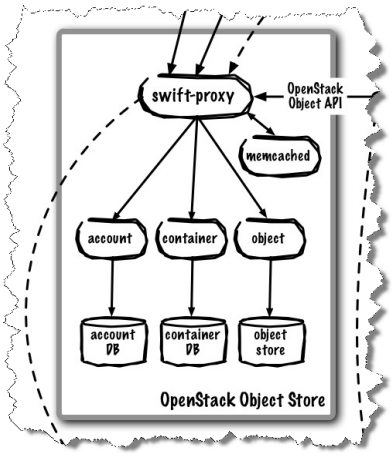
OpenStack Object Storage – Swift

As one of the two core projects since the inception of OpenStack, Swift handles object storage. A challenging concept for some who come from traditional server block storage environments, object storage is a highly scalable storage environment which can contain photos, backup images, virtual machine images and other static content.

The architecture of Swift storage uses distributed storage across multiple disk drives and provides redundancy and protection by storing multiple instances of each object across different drives. The JBOD (Just a Bunch of Disks) storage methodology allows for simple scale-up because we can simply add new disks which are added to the storage environment and the cluster is expanded using the newly available storage without having any need for deep awareness of the underlying storage infrastructure.

This is a phenomenal example of the use of commodity storage because of the nature of the content and the limited need for high performance because objects are written once and then only serve up read requests from that point onwards.

Swift uses an API and a proxy to handle requests and send the reads and writes to the appropriate storage nodes based on a number of criteria. The compute resources always use the Swift proxy to access the objects which follows the model of abstracting the layers to prevent the rigidity of directly connected functions that require an awareness of path and failover capability. Using the Swift API, the requesting resource has no awareness at all of which location will serve up the request.



INSERT 9376OS_01_09.PNG

The Swift component also introduces the concept of rings, which we will discuss at length in a later chapter. The ring structure is the heart of the distributed storage capability and is a powerful feature that can handle incredible amounts of data.

Swift is able to handle single object size up to 5GB, but for larger objects the product uses segmentation to store 5 GB “chunks” while storing the configuration in a single manifest about the object. When the object is downloaded, the manifest is used to recreate the structure and concatenate the object segments into a single object for retrieval.

OpenStack Image Service – Glance

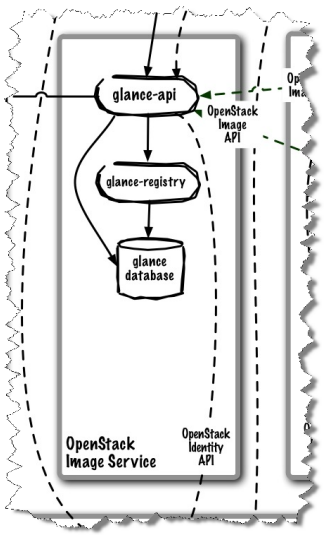
Not to be confused with images in the sense of pictures, this is where virtual machine images are stored which will be used for generating instances of your virtual servers. The objects are held in our storage environment as objects and can be stored in a variety of formats.

As with any of the OpenStack components, the goal is to support and promote the uses of open standards for better interoperability. The architecture of Glance has also been built to leverage fault tolerant features to create high availability deployments.

Images can be used from a variety of sources, and as we discover later in the book, creating your own images is as simple as your current template creation in whatever your virtualization platform of choice is.

Storage of images can be done in a traditional file system, as objects in your OpenStack Storage environment, in the public cloud using Amazon S3, and you can also retrieve images over HTTP which enables a wide variety of options for you. As we delve into the details of each type of storage method, we will evaluate the advantages and challenges of each type.

Many image types are supported including Raw, qcow2, vmdk, VHD and OVF among others depending on your hypervisor of choice. This flexibility will make the image creation process easy to adapt to for your administrators.



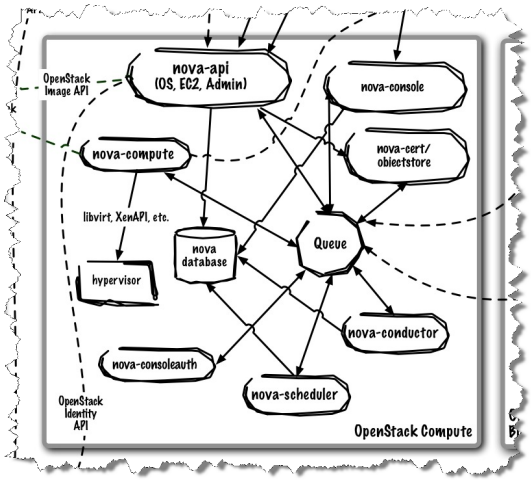
INSERT 9376OS_01_10.PNG

OpenStack Compute – Nova

The heart of the OpenStack environment is the Nova compute component. The default configuration of Nova uses the KVM compute architecture. For those coming from a Linux background, KVM is a familiar product and is a versatile compute infrastructure.

Additional hypervisor support is available for Xen, QEMU, LXC, ESX, Hyper-V, Baremetal and PowerVM although the supported features vary from platform to platform. For a full list of supported features, please go to https://wiki.openstack.org/wiki/HypervisorSupportMatrix to see the updated matrix.

The Nova project contains all of the control and orchestration features, but does not contain the hypervisor itself. All of the drivers for the supported hypervisors are delivered in the Nova core. Because we are working with the Ubuntu deployment for most examples in the book, a significant amount of the samples will use KVM but we will explore additional hypervisors also along the way.



INSERT 9376OS_01_15.PNG

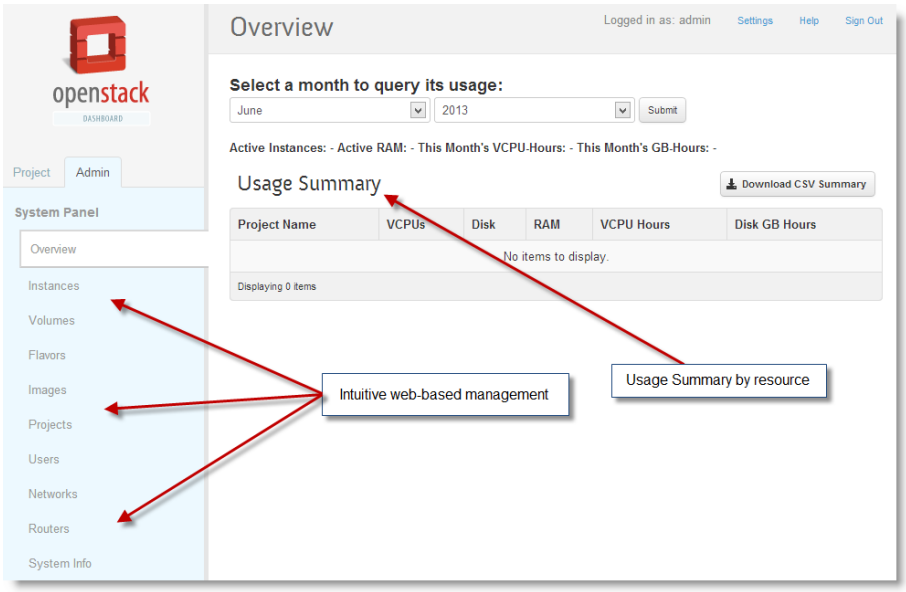
Beyond the hypervisor management, Nova handles VLAN, IP address allocation, projects and quotas. The Nova networking components may be used in absence of the OpenStack Networking (Neutron) component, but there will be much more discussed later on this subject.

OpenStack Dashboard – Horizon

In order to deliver services to our consumer, we use Horizon Dashboard. The cleanly designed and fully featured web console is where users and administrators will view and manage their instances, storage and security.

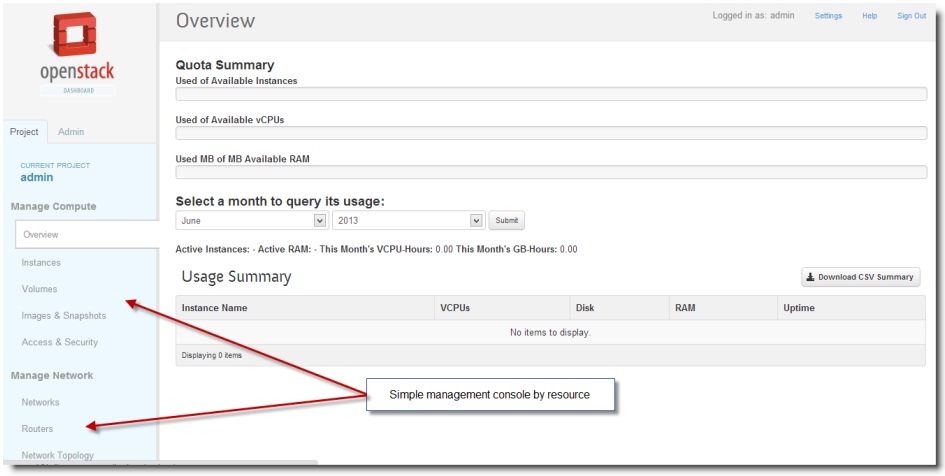
The admin features of the OpenStack dashboard include the ability to create, manage and remove every component of an OpenStack environment including:

- Users
- Projects
- Networking
- Resource limits



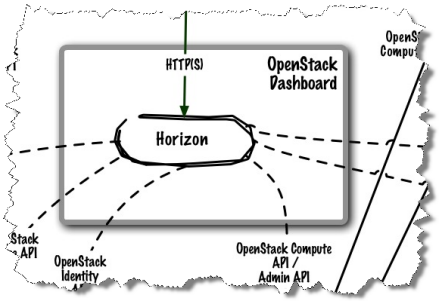
INSERT 9376OS_01_12.PNG

As a consumer of the service, the OpenStack dashboard is your complete self-service portal for deploying and managing your compute, storage and networking resources as defined by your administrator.



INSERT 9376OS_01_12.PNG

The dashboard can be easily customized and branded using simple HTML and CSS. With the use of the well-documents APIs for all of the OpenStack components, the extensibility of dashboard and web components is an option for those organizations who wish to create distinct, custom interfaces if they wish to.



INSERT 9376OS_01_13.PNG

The configuration of the Horizon dashboard is surprisingly simple, but its features and functionality are significant. This is the portal for the consumer and the administrator for a large portion of the day-to-day use of your OpenStack environment.

OpenStack Networking – Neutron (Formerly Quantum)

What was originally known as nova-network has expanded to become the OpenStack Networking project. At the time of the Grizzly release, the codename was Quantum. Due to issues of this conflicting with an existing trade name, the project is being renamed to Neutron.

It is possible to use the traditional nova-network components without Neutron, but in order to create advanced networking layouts, we will be using Neutron with a variety of vendor supported plugins to leverage industry standard implementations and work with your existing infrastructure using best-of-breed capabilities.

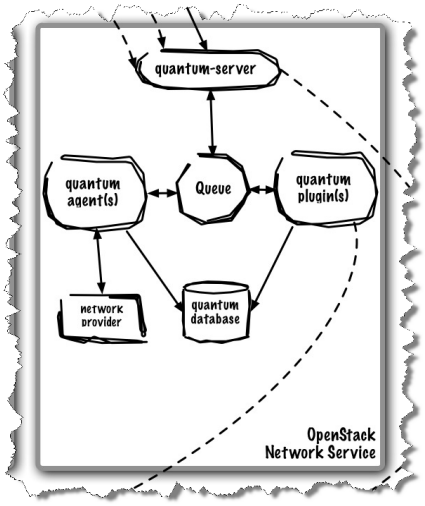
Network virtualization is achieved by managing L2-L7 of the OSI stack using software based management, which allows us to standardize, automate and

decouple from the hardware layer of the network. While we still rely on L1 hardware, the abstraction of networking through virtualization removes the rigid dependency on hardware and vendor lock-in. Additional gains are made by utilizing vendor plugins which can still leverage the L1 features of the hardware, which makes our network virtualization as fully interoperable as possible.

Architecturally, the OpenStack Networking is comprised of a few core services:

- Plugin agent – the vswitch configuration is set up for each hypervisor
- DHCP agent – DHCP services for your tenant networks
- L3 agent – Layer 3 access for your tenant networks

Plugins are available for a number of vendors, and more are being developed today for future support. This is a key feature to the OpenStack Networking project, because it enables customers to leverage current vendors (hardware and software based) to get the best interoperability for your OpenStack deployments.

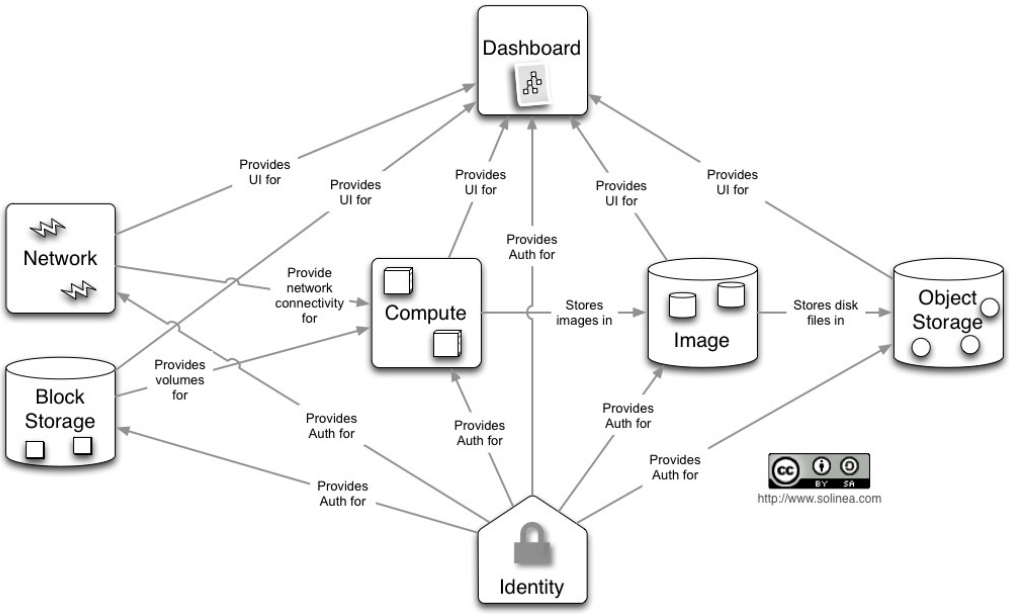


INSERT 9376OS_01_14.PNG

This will be one of the challenging subjects to approach for many as there are many ways to design and deploy your tenant networks, so we will spend some time getting much deeper with this in chapter 8.

How Does it All Work Together?

As important as seeing the individual projects, we have to understand the workflow and how the components interact and exchange data during operations. Let's quickly look at the diagram again to see how the projects come together.



INSERT 9376OS_01_16.PNG

The lines flowing between the systems indicate the actions being performed by the connecting system. As you can see, there are two key components which are always present which are Identity (Keystone) and Dashboard (Horizon) which perform actions of "Provides Auth for" and "Provides UI for" respectively.

As we dive into each of the components in the coming chapters, you will understand better how the workflow between components goes with each specific activity being performed in the OpenStack environment.

Most importantly, you can see the dependency on the interaction between the components. Our goal is to fully understand how and why these interactions occur as we continue our journey in to Mastering OpenStack.

Incubation Projects

As mentioned earlier, there are projects which are referred to as incubation projects. There are a few that are in progress at any given time, but two that I wanted to highlight in particular ones that will most likely be heavily featured heading into the next release of OpenStack (Havana).

Metering - Ceilometer

A key feature of a cloud framework is the ability to manage utilization and generate analytics on performance. The Ceilometer project will be integrated with future releases as the metering engine to provide a one-stop-shop for analytics and usage information.

The goal as stated on the project wiki is to “It's primary targets are monitoring and metering, but the framework should be easily expandable to collect for other needs.” (Source: https://wiki.openstack.org/wiki/Ceilometer).

We will not be covering Ceilometer as a part of the book as it is not fully deployed with the Grizzly release.

Orchestration - Heat

Based on the AWS CloudFormation template format, the Heat project is being designed as an orchestration engine to automate the deployment of OpenStack resources (compute instances, volumes, floating IP addresses and more).

Deployment of resources in traditional format as well as high availability and scale-on-demand is the ultimate goal. This will create a very developer-friendly platform which has been what holds many developers in the AWS environment.

Included in the project is the heat CLI (python client), a RESTful API, and the heat engine which is handles the orchestration and event notification. As mentioned in the wiki, this has great potential to be integrated with Puppet and Chef.

We will not be covering Heat as a part of the book as it is not fully deployed with the Grizzly release.

Distribution Support for OpenStack

At the time of this writing, there is support for OpenStack distributions on multiple platforms which include:

- Ubuntu Linux
- OpenSUSE Linux
- SUSE Linux
- Debian Linux
- Fedora Linux
- Red Hat Enterprise Linux
- Cloudscaling
- StackOps (Nova)
- SwiftStack (Swift)

Additional distributions will inevitably become available as the OpenStack ecosystem and supporting partners grows. At this time, there is no OpenStack certification program for supported operating systems.

As mentioned earlier in the chapter, the examples within the book are mostly built on an Ubuntu Linux platform, so some distribution specific features and idiosyncrasies may come into play. The focus of the book is on the OpenStack APIs, clients, features and configuration, so the distribution is less important for the content.

Some challenges come with Linux systems and software, and also with the OpenStack software itself because of the active development of these platforms. There will be patches released to both the Linux products and to the OpenStack software, so we have to be mindful of that. This is another reason that we put the focus on the operational use of OpenStack rather than the installation and baseline configuration.

Chapter Summary

We have covered a lot of general information in the chapter. Our goal is to ensure that we have a full understanding of the general cloud concepts, the basics of the OpenStack projects and a high-level view of how they interact.

Understanding the use of OpenStack APIs, the RESTful concepts and the project features will be an ongoing theme throughout the book. The detail of the foundation, contributors and how Grizzly has come to be will be helpful as we move forward.

If you are a newcomer to the OpenStack environment, this is where things will ramp up quickly, but the goal is to lay the foundation to start with the overall logical structure and then we will drill down as we push forward into the next chapters.

For those who have already had experience with OpenStack, this will be a review, so don’t worry, we are heading into the fun stuff next!