PART I
OpenStack Overview

1 Introducing OpenStack

WHAT'S IN THIS CHAPTER?

- ➤ Models of cloud computing
- ➤ Relevance of cloud computing to application developers
- ➤ Why OpenStack is a good cloud platform choice
- ➤ How OpenStack is put together

WHAT IS CLOUD COMPUTING?

There is so much hype around cloud computing that it is often difficult to get a clear sense of what anyone means by those words. Is it just virtualization? Is it *Software-as-a-Service* (*SaaS*), such as Microsoft's Office 365 and Salesforce.com? Or is it the ability to get a virtual machine instantly from Amazon Web Services (AWS) or Azure? And what about online storage such as Dropbox?

Types of Cloud Computing

The reality is that cloud computing refers to all of these things just described and more. The National Institute of Standards and Technology (NIST) has come up with an "official" definition based upon five key components: on-demand self-service, broad network access, pooled resources, elasticity, and metered service. In general, these characteristics may be provided in several different models. These models help sort out the confusion and hype. In fact, these can be thought of as layers in a stack, with each layer being built on top of the previous one (see Figure 1-1).

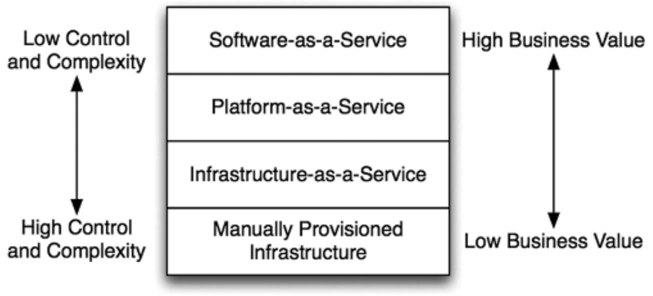

Low Control and Complexity
High Control and Complexity
Software-as-a-Service
Platform-as-a-Service
Infrastructure-as-a-Service
Manually Provisioned Infrastructure
High Business Value
Low Business Value

FIGURE 1-1

In Figure 1-1, "Manually Provisioned Infrastructure" represents the traditional method of building your information technology infrastructure—this is not cloud computing. In this environment, physical machines are racked, connected, and configured on a one-by-one basis. This provides complete control, but requires substantial time and effort to build out, or to change when necessary. Of course, all clouds need to run on physical gear at some point, so this provides the basic foundation for everything else. One of the keys to making cloud computing successful, however, is to move the complexity out of this layer and up higher in the stack.

Infrastructure-as-a-Service (*IaaS*) is the most basic layer in the cloud computing stack. This is OpenStack's primary focus, as well as the primary focus for AWS. It enables automated or self-service provisioning of compute, networking, and storage. Typically, these resources are provided as Virtual Machines (VMs), but you could also use it to spin up bare metal servers (i.e. physical hosts). This is known as "Metal-as-a-Service," and OpenStack provides a project for managing this service as well. Alternatively, you can also spin up containers rather than VMs or bare metal servers. The essential point is that it enables the provisioning of compute instances, with (optionally) attached networking and storage.

Platform-as-a-Service (*PaaS*) builds on top of IaaS to enable the provisioning of applications, rather than simply the infrastructure that might be used to run the application. So, a PaaS provides core common services needed by applications, along with the machinery to configure and deploy applications to use those services. A PaaS typically will provide a complete application stack (web server, application server, database server, etc.) into which you can easily deploy your application. Heroku (`https://www.heroku.com`) is an example of a popular PaaS for applications built with a variety of standard frameworks, such as Ruby-on-Rails. With Heroku you can deploy your application to the Internet with a simple `git push`. As the application author and deployer, you don't need to worry about configuring and deploying the different tiers, or even worry about how to scale them. If you follow the Heroku conventions, everything is handled by the PaaS.

Software-as-a-Service (*SaaS*) is the layer farthest from the underlying physical infrastructure. It may be built on IaaS or a PaaS, but need not be—the point is the user never really knows. This is the simplest form of cloud computing from the point of view of the user because they have no insight into the actual mechanics or systems behind the service. It's just a service they use. Often this is provided in the form of a website, such as Salesforce.com. But you can also get lower-level services such as Database-as-a-Service, where you simply request via an API (or website) for a database with certain parameters, and are given an IP and port to connect to. As a user of the service, you don't need to worry about how to scale that service—though you will need to pay more as your use of the service increases.

Put succinctly, IaaS provides the tools to "build" your systems from the ground up. PaaS allows you to "deploy" your applications, without needing to worry about the underlying infrastructure. SaaS allows you to "buy" your applications—you do not even need to deploy or manage them at all. This is a steady progression of decreasing control and complexity, while increasing direct business value.

While these are general models for cloud computing, in reality the distinctions between them are not always crystal clear. The relationship of SaaS to PaaS in particular can be complicated. A specific, complex Software-as-a-Service may use PaaS or even other more granular Software-as-a-Service. Even a PaaS may assemble lower-level pieces as a collection of software services. For example, most services will require an identity management (authentication, authorization, and accounting) service. This identity service is one of the key features a PaaS provides to applications. However, there is no reason that service cannot be, in turn, provided by some external SaaS! In this case, a key function of the PaaS is provided via a low-level SaaS.

Cloud Infrastructure Deployment Models

In addition to the functionality provided by a cloud, there are several different deployment models for clouds. *Public clouds* are the ones familiar to most developers. These cloud services are made available to the general public for a fee. The fee is generally on a usage basis, enabling organizations to utilize their operating budgets rather than their capital budgets. The customers have no need to maintain or operate the hardware or cloud infrastructure, leaving that responsibility completely to the cloud operator.

Amazon Web Services (AWS) is currently the largest public cloud and dominates the industry. Microsoft and VMware also operate public clouds, and a number of service providers do as well. Rackspace, in particular, provides an OpenStack-based public cloud, and is one of the primary contributors to the OpenStack project.

Private clouds, on the other hand, are internal to an organization. They represent the evolution of the traditional corporate data center. Only internal customers within the enterprise, and perhaps close partners, use private clouds. The corporate IT department or a contractor will purchase, setup, and maintain the hardware and software for the cloud. The cloud infrastructure may use chargeback to distribute costs among the business units, but the cloud itself is still dedicated to the single enterprise.

Organizations may operate private clouds for a number of reasons. The cost of a private cloud, if well run, may be less than utilizing the public clouds. Additionally, many industries have security or regulatory reasons that disallow the use of a public cloud for many workloads. These organizations are required to run those workloads in a private cloud. See Figure 1-2 for a look at the structure of public, private, and hybrid clouds.

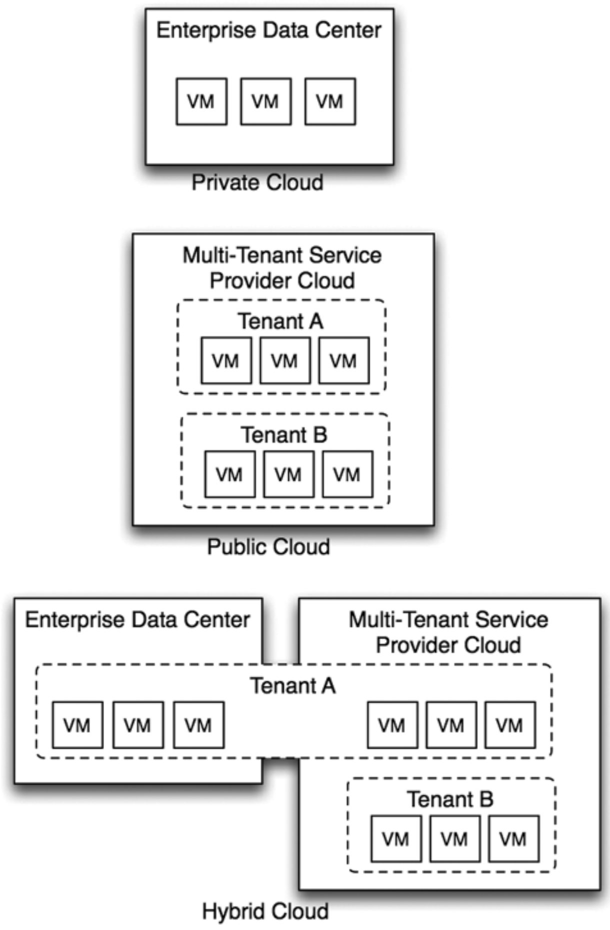


FIGURE 1-2

Hybrid clouds combine both private and public clouds. The goal with hybrid clouds is to keep general operating costs low by using the private cloud for most of the workloads, but to enable spillover into the public cloud when necessary. The spillover could happen due to capacity reasons—perhaps during the holiday season your private cloud doesn't have enough capacity—or for disaster recovery. This model avoids the capacity constraints of a private cloud while still keeping costs under control.

WHY SHOULD I CARE?

As an application developer or architect, you may wonder—why does all of this matter to me? All of this discussion covered so far focuses on the reason a business may want to move to the cloud. But why should that affect the application developer? The answer lies in a couple of different areas: the effect on the development process, and the effect on your application architecture.

Cloud services enable much more efficient processes for managing development, test, and production environments. These updated processes and methods represent the “DevOps” mentality—applying standard software development practices, such as source code version control, to the operational aspects of the application. This means capturing all of the configuration and deployment information in scripts and templates, and controlling their changes just as you would application code.

Scripts and templates can be built that produce a complete application environment. These can be used to automatically deploy not only the application, but also infrastructure required for the application, including virtual machines, networking, firewalls, load balancers, domain name services—you name it, and someone is working on making it available “as-a-Service.” By automating the creation and destruction of these environments, you can ensure consistency between development, test, and production environments. For complex applications with many different services running on different machines, this can be a dramatic time saver.

OpenStack, and “as-a-Service” thinking in particular, will also end up changing the software and deployment architectures of your application. By relegating the common and routine functions to the cloud infrastructure, you free your time and thought to focus on the most important thing—your application’s functionality. For example, a traditional application that allows large file uploads will need to designate temporary and permanent storage locations for those files, and manage the storage resources to ensure that the disk doesn’t fill. The system administrator or deployer will need to devise a strategy to backup that data or replicate it to other data centers. But with the right cloud platform, you can simply delegate that function to the infrastructure, and get all of the benefits without devoting special effort.

Designing your application to work with the cloud services also dramatically simplifies scaling the application. The scalability of the individual services becomes the responsibility of the cloud operator, not the application developer or administrator. As long as the application makes effective use of those services, it will scale as needed with little to no work from the developers themselves.

Being able to utilize “as-a-Service” functions is one way your design will shift. Another is to plan for horizontal scaling rather than vertical scaling. That is, scaling by adding more machines (horizontally) rather than creating bigger machines (vertically). With most applications today, it is easiest to scale by getting a bigger, faster machine. This locks you into planning for peak capacity of each application individually. For each application you need to provision the largest machine you may need at peak load. But with applications built for the cloud, you instead scale by adding more machines. These machines can be smaller, and with cloud automation, can be added, removed, or resized as needed. This ability to scale up and down as needed is called *elastic scaling*, and is one of the key features of cloud computing.

A frequently used analogy is that traditional servers are like “pets,” while cloud-based servers are “cattle.” This describes a necessary shift in mentality for a traditional application architect. The idea is that a pet is unique and special, with its own unique name. A lot of resources are spent to raise and nurture one, and if it is sick, it will be nursed back to health. Cattle, on the other hand, are not treated specially or carefully raised. They are treated en masse—they are given numbers, not names—and a sick one is culled to prevent any spread of disease through the herd.

The implication here is that cloud-based servers should be disposable and easily re-deployed, and not require careful hand configuration. That way, if there is a problem with one, you do not spend time trying to figure it out and fix it—you simply replace it with a new one. This is the logical extension of the ability to scale elastically. Why take the time to figure out what's wrong with a machine when it's behaving badly? Just pull it out of the application and replace it with a new one while you debug the problem (not to fix that machine, but to prevent the issue in the future).

What Is OpenStack?

OpenStack bills itself as a "cloud operating system." Fundamentally, it solves the IaaS problem. It provides the ability to abstract the physical compute, storage, and networking resources into pools. Those resources can then be divvied up among users in a secure way. Users only need to pay for what they are using, rather than having to provision their applications for peak load.

OpenStack is a collection of open source software projects, backed by a non-profit organization, the OpenStack Foundation. These projects work together to provide a consistent API layer, while enabling the actual services to be provided by a variety of different vendor or open source implementations. At the core, these services include the functionality you need to run a cloud, that is, the ability to spin up virtual machines, the ability to allocate, manage, and share storage among those machines, and the ability enable these machines to communicate with one another securely over the network.

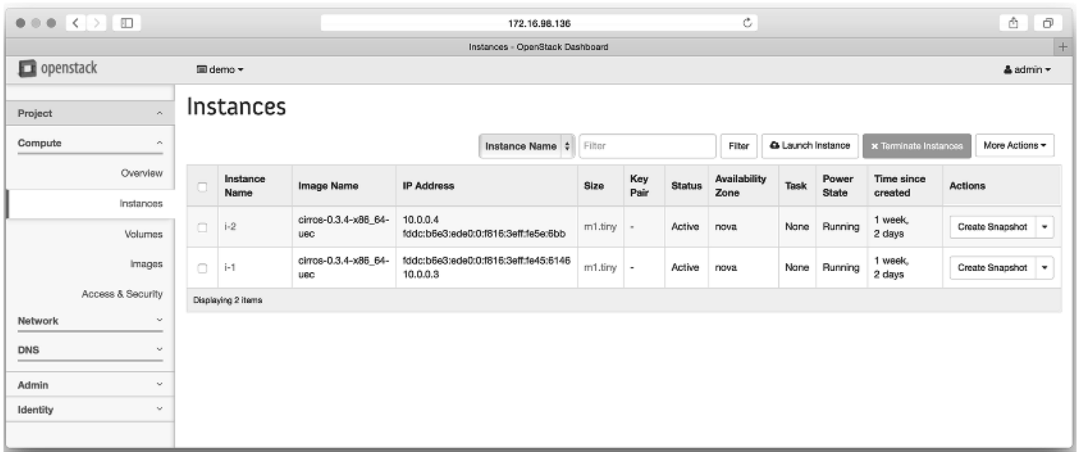
KEEPING TRACK OF RELEASES

OpenStack has official releases every six months. In order to make it easier to keep track of all these releases, they are given names in alphabetical order. Below is the name of each release, and its release date, through the Liberty release.

- Austin: October 2010
- Bexar: February 2011
- Cactus: April 2011
- Diablo: September 2011
- Essex: April 2012
- Folsom: September 2012
- Grizzly: April 2013
- Havana: October 2013
- Icehouse: April 2014
- Juno: October 2014
- Kilo: April 2015
- Liberty: October 2015

In addition to the release name, each release is identified by the year and release during that year—<year>.<release>.<patch>. For example, Kilo is also known as 2015.1, as the first release in 2015. Patch releases for Kilo are 2015.1.1, 2015.1.2, etc. The second major release of 2015 is Liberty, which is also known as 2015.2.

All of these services are accessible via RESTful APIs, as well as command-line interfaces and a web-based user interface called Horizon. Horizon is convenient for setting up things on an ad-hoc basis, but doesn't offer the full capabilities of the APIs—and of course the APIs and CLI tools can be easily scripted (see Figure 1-3).



	Instance Name	Image Name	IP Address	Size	Key Pair	Status	Availability Zone	Task	Power State	Time since created	Actions
	i-2	cirros-0.3.4-x86_64-uec	10.0.0.4 fddc:b6e3:ede0:0:f816:3eff:fe5e:6bb	m1.tiny	-	Active	nova	None	Running	1 week, 2 days	Create Snapshot
	i-1	cirros-0.3.4-x86_64-uec	fddc:b6e3:ede0:0:f816:3eff:fe45:6146 10.0.0.3	m1.tiny	-	Active	nova	None	Running	1 week, 2 days	Create Snapshot

Displaying 2 items

FIGURE 1-3

The next table shows the major services provided by OpenStack, along with their names. OpenStack community members will usually refer to each service by its name, so it's helpful to see them all in one place and get a handle on what each one does. In fact, there are many more services, but these are the most common ones you will find.

NAME	SERVICE	DESCRIPTION
Horizon	Dashboard	A graphical user interface for managing your cloud
Keystone	Identity	Authentication, authorization, and OpenStack service information
Nova	Compute	Spin up, manage, and terminate virtual machines
Cinder	Block Storage	Disk volumes (that outlive an instance) and snapshots of instances
Swift	Object Storage	Shared, replicated, redundant storage for images, files, and other media accessible via Hypertext Transfer Protocol (HTTP)
Neutron	Network	Provide secure tenant networking
Glance	Image	Provide storage and access to VM images and snapshots
Heat	Orchestration	Spin up groups of machines, networks, and other resources via templates

continues

(continued)

NAME	SERVICE	DESCRIPTION
Designate	DNS	Create domains and records in the DNS infrastructure
Ceilometer	Telemetry	Monitor resources usage across the cloud
Trove	Database	Provide access to private tenant databases
Ironic	Bare Metal	Spin up instances on physical hardware
Magnum	Containers	Manage containers within instances
Murano	Application	Deploy packaged applications across multiple instances
Sahara	Data Processing Cluster	Provides a Hadoop or Spark cluster as a service

A default installation of OpenStack will include "reference" versions of each service. For example, by default an OpenStack cloud will use the Kernel-based Virtual Machine (KVM) hypervisor to manage virtual machines. One of the most important aspects of the OpenStack architecture, however, is the driver or plugin-based nature of each service. With this design, you can use an implementation other than the reference one. In your cloud, you can swap out KVM with ESXi, Xen, or other hypervisors. The APIs used to launch and manage VMs remain the same, regardless of the underlying hypervisor. This same concept extends across OpenStack services, enabling the same APIs with different service implementations.

This level of flexibility behind the scenes, while providing a consistent API, is one of the keys to the success of OpenStack. Users can build their applications and automation on top of OpenStack, without having to worry that they are locking themselves into a single backend provider of computer, networking, or storage. The APIs won't change even if they swap out the backend.

OpenStack is frequently used in enterprises for private clouds, though there are some public cloud services that are based on it. There are also companies that will create and operate a private OpenStack cloud for you within their data centers. In this case, the hardware is not shared with other customers, so you have the predictability and security of the private cloud but do not have to find and hire the experts to maintain it.

Even in private cloud environments, OpenStack is a *multi-tenant* cloud platform. This means that multiple users or groups of users—*tenants*—can utilize the physical resources of the cloud, while keeping all of their virtualized resources private. For a tenant, the OpenStack environment appears, for the most part, to be theirs and theirs alone. But for the operator, the underlying physical resources and software systems are shared. In OpenStack, tenants are also sometimes referred to as *projects*.

In a multi-tenant OpenStack cloud, each tenant is allocated a *quota* for the various types of resources that may be used. The quota provides a maximum limit for that tenant for that particular resource. You will have a quota for CPUs, memory, storage, networks, subnets, and floating IPs, among other resources. This prevents any single tenant from consuming all of the resources.

Why OpenStack?

There are a number of cloud management platform options out there. The most obvious and dominant player is VMware with their vRealize suite of software. So, why should you take your time to learn about OpenStack rather than vRealize, AWS, Azure, CloudStack, or any of the other solutions?

About 15 years ago, IT professionals faced a very similar set of questions about Linux and proprietary UNIX systems. Solaris, HP-UX, AIX and their competitors were solid, well known, and widely deployed products, whereas Linux was a graduate student's project that was difficult to install and operate and was fairly immature, with driver and other compatibility issues. It was not clear at all at the time that spending effort learning and understanding Linux was worth it. History though, has proven that such a choice would have been the right one. All of those expensive, proprietary UNIX implementations have lost their value proposition—they really don't have much that is unique to offer anymore. Linux has continued to grow and has taken over most of the environments where those systems once thrived.

This isn't just a simple analogy. There is a relentless pressure in this industry to reduce costs, and to increase the velocity of feature delivery—deliver more, faster, and cheaper. The way to achieve "more, faster" is standardization. This is the same basic principle as building libraries and frameworks in programming. A standard architecture behaves in a predicable manner, providing core services on which you can rely and build. There is no need to repeat the process of developing that architecture over and over, allowing you to focus on the new functionality.

The way you achieve "cheaper" is to make those standards open and free. This combination of open and standard leads to *commoditization*—essentially the development of interchangeable components that are the same regardless of the manufacturer or vendor. Commodities imply a lot of competition, and there is little or no product differentiation for which to charge extra. This drives down the costs dramatically.

Linux has both of these characteristics—open and standard—in UNIX-like operating systems, and that is why it won. Not because it was better, but because it was cheaper and faster to use as a base for building new functionality. Linux is just one example, of course. This story has repeated over and over in the technology industry. With machine architectures we have the x86 platform, and standard architectures for memory, disks, and serial bus-based peripherals.

In fact, if you take the broader view, you can see that the commoditization has continuously moved up the value chain. It started with hardware, moved to operating systems, and these days even sophisticated databases and distributed system components are being commoditized. In databases, we used to have Informix, DB2, Oracle, Sybase, and others. But MySQL and PostgresSQL are open and standard, and they have completely dominated the low-end of the database market. Oracle still leads in the high-end, and is able to provide value in those more specialized environments, but as the open source products improve, the space for the proprietary vendors constricts.

In some way, cloud computing is the culmination of this commoditization process in the industry. Broadly, you can think of the revolution happening in the computing industry as a refocusing of the industry on the core functions of computing. The abstraction of the computing infrastructure into simply compute, storage, and networking components, and breaking of these out from being vertically integrated, to horizontally integrated, is truly transformative. It brings full commoditization to these elements, which are the basic foundation of the industry.

Cloud platform management will follow the same pattern. The proprietary platforms like vRealize will thrive for a time, but in the long run the open and standard systems will win. While there may always be a place for the proprietary solutions in more specialized environments, the most common platforms will be open source. You can see this already happening: the Zenoss 2014 State of the Open Source Cloud Survey (`http://www.zenoss.com/resource-center/white-papers`) found that 30 percent of respondents were already using an open source cloud, up 72 percent from 17.2 percent in 2012. Another 34 percent of the respondents planned to implement an open source cloud in the future. Understanding this gives you an advantage to focus on the eventual winner, instead of chasing what will ultimately be a setting star.

There are several open, standard cloud management platforms. So even if you believe that the bet on open and standard is the way to go, why should you bet on OpenStack? The answer here is simple—momentum. OpenStack is by far the most widely used and supported open source cloud management platform, and it has the largest community of developers and vendors push it forward. The same survey mentioned above found that 69 percent of respondents with an open source cloud were using OpenStack in 2014, up from 51 percent in 2012. An amazing 86 percent of those considering an open source cloud deployment are looking at OpenStack.

The OpenStack developer and user communities have grown dramatically as well. The OpenStack Foundation 2014 Annual Report (`https://www.openstack.org/assets/reports/osf-annual-report-2014.pdf`) provides detailed insight into this growth. In 2013, the best quarter for mean monthly active developers had 391 developers—in 2014 this measure was up 45 percent to 569 developers. Large investments from HP, Cisco, RedHat, IBM, Dell, Mirantis, Rackspace, and many other vendors have driven this surge. The incredible growth in the number of users, developers, and other interested parties can be seen from the attendance at the twice annual OpenStack Summits, seen in Figure 1-4 (source: openstack.org).

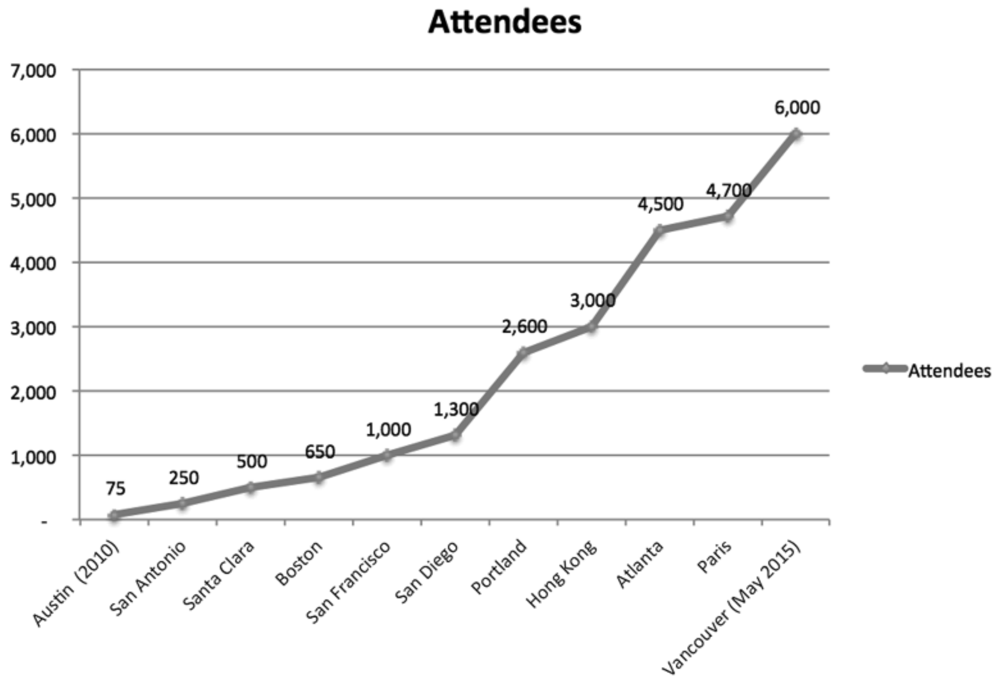


FIGURE 1-4

Clearly OpenStack has the momentum to succeed.

UNDERSTANDING THE ARCHITECTURE

OpenStack is built on a loosely coupled architecture. Each component is built independently and runs its own services. These services may be distributed among a number of different machines with different responsibilities. This enables scaling of particular functions, by adding machines with particular roles. It also enables redundancy; a highly available deployment will contain several of each type of machine.

Software Architecture

Individual components interact with one another via well-defined application programming interfaces (APIs)—typically based on representational state transfer (REST) conventions, though in some cases using remote procedure calls (RPC) or notifications over a message bus. Typically, these services will maintain data in a relational database—usually MySQL or PostgreSQL. The message bus and database are shared across services, but the interactions between those services remain clearly delineated. This enables different services to grow and change independently from the others, so long as they provide backward-compatibility in the APIs.

Each of the major services—compute (Nova), networking (Neutron), block storage (Cinder), etc.—have several internal processes and components. Generally, they will each have an API service that provides an HTTP-based RESTful API. This API service will communicate with the other components via the message bus.

The Horizon service is a web-based UI that interacts with the various services. Similarly, there are command-line tools to interact with each service. These tools are optional; you can build your own interface directly to the service APIs if you wish. Horizon and the official CLI clients do not have any special access; everyone uses the same APIs. Each client really only needs to be informed of the location of Keystone, the identity service. This service contains a catalog of all services and API endpoints available in the OpenStack platform (see Figure 1-5).

In Figure 1-5, what you see is a simplified depiction of how the services interact. Each service has an API component, which communicates with Keystone's API via HTTPS to provide authentication and authorization information. Each API service uses the message bus to communicate with several other processes for that service (just called "Services" in the diagram). As needed, these downstream service processes will call the APIs of other services. For example, Nova will call the Neutron API to acquire a port on a particular network.

Deployment Architecture

How are all of these different pieces of software deployed on the hardware? This is actually pretty flexible. For development or just experimentation, you can even run everything on a single machine. However, a more typical deployment will have several controller nodes (for high availability purposes), along with additional network, compute, and controller nodes.

Each high-level service (compute, networking, storage, and others) consists of multiple daemons (background processes). These daemons are spread out across the various types of nodes. That is, you do not run individual services on individual nodes, but rather spread each service out across different types of nodes.

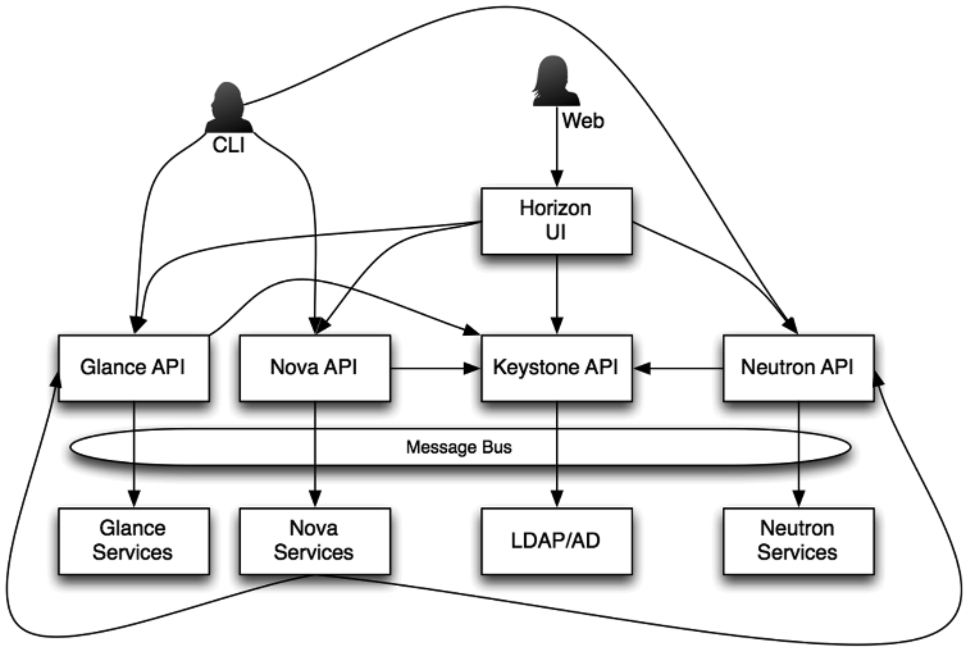


FIGURE 1-5

For example, all of the services share the database and messaging components (typically MySQL and RabbitMQ, respectively). You may run these each on separate clusters, with each cluster spread over different failure domains. Additionally, you may have several physical nodes that provide the API endpoints, behind a physical load balancer. Different daemons for Nova and Neutron will be spread across the network and compute nodes. Figure 1-6 shows a simplified diagram of this layout.

Notice the different types of nodes in Figure 1-6. *Compute nodes* run the hypervisor and therefore the actual VM instances, as well as provide the ephemeral storage for instances. They will also run Neutron networking agents to manage the connectivity between VMs (called *east-west traffic*).

The *network nodes* usually provide the connectivity between VMs and outside the cloud (called *north-south traffic*), as well as the advanced network services like load balancing and VPN access. Depending on the choices made by the administrators and users, there may be agents providing network routing services on the network nodes, directly on the compute nodes, or both.

The *block storage nodes* provide volume services to the instances—that is, they provide access to persistent storage for disk volumes that can be attached and detached to instances. Clouds that offer *object storage* will also have separate clusters for that. Object storage provides shared, replicated, redundant storage for images, files, and other media accessible via HTTP.

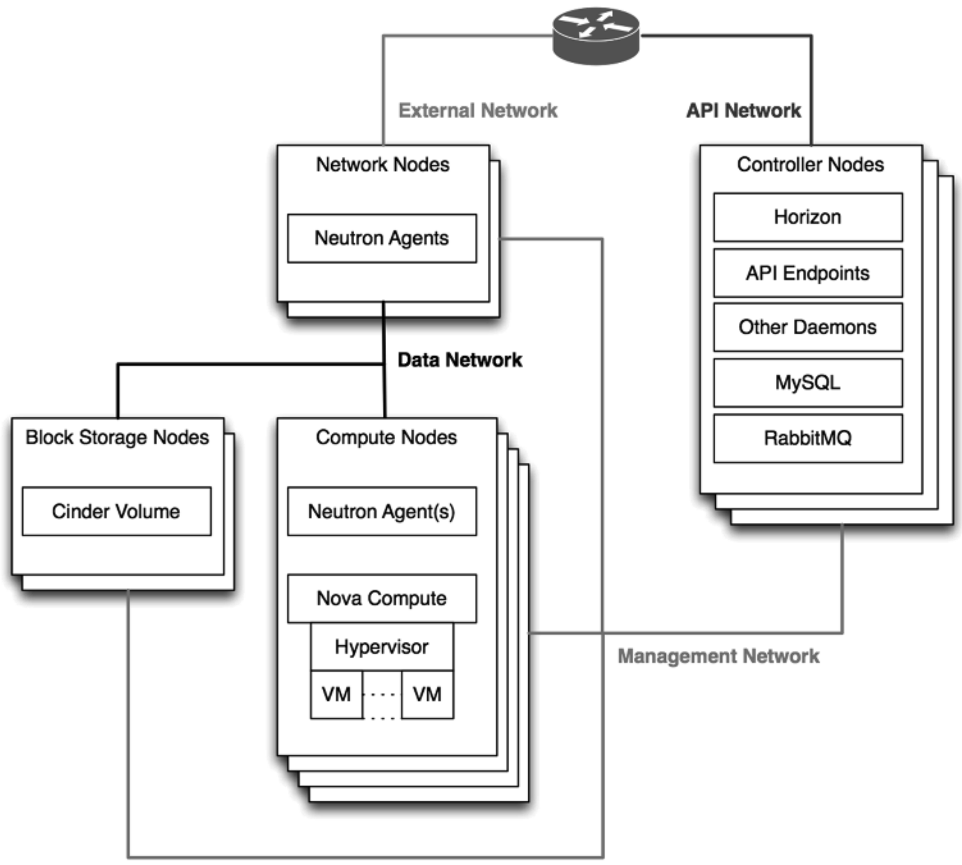


FIGURE 1-6

Various segregated networks connect all of these nodes. Every node is accessible via the *management network*, which is used for different parts of OpenStack to communicate with one another. All of the message bus, database, and cross-project API traffic go over the management network. The *data network* connects all of the compute nodes, network nodes, and block storage nodes. The internal cloud tenant traffic is carried on this network, whereas the *external network* provides access to the outside world. Since the compute nodes do not communicate with the outside world, but only with other nodes in the cloud infrastructure, they need not have connectivity to the external network, but only need access to the data network. Only the network nodes need to connect to the external network. Finally, some installations will use an *API network*, which provides access between the outside world and the OpenStack end points (API and Horizon), separate from the external network used by tenants.

Pros and Cons

This architecture provides a great deal of flexibility. This enables the scalability by letting the cloud operator deploy additional nodes to scale the infrastructure. It also allows the ability to create highly available services, since you can split each service out and make them redundant across failure domains. However, it is very complex, and can be quite difficult to set up and maintain.

As a user of the cloud, this will be transparent to you. But a properly run cloud will have enough redundancy built in that you can expect a high-level of reliability from the OpenStack infrastructure.

Another substantial benefit to this architecture is avoiding vendor lock-in. Each service provides a plugin or driver-based architecture. This enables each service to work with any number of vendor platforms to provide the actual service. For compute, you can use the default KVM hypervisor, ESXi, Xen, or one of many other hypervisor choices. The networking service defaults to using Open vSwitch to provide Layer 2 (the data link or MAC address layer) connectivity, and the Linux networking stack (iptables, routing, and namespaces) to implement Layer 3 (IP layer) functionality. However, there are more than 20 different vendor plugins to swap all or part of that default implementation. In fact, these vendor implementations can be used at the same time, in the same cloud.

By avoiding vendor lock-in, OpenStack enables more competition between the vendors, pushing down prices in the market. The ability to use multiple vendors at once makes transitioning from one vendor to another more feasible, and also allows the choice of vendor for solving specific use cases.

An interesting feature introduced with the Kilo release of OpenStack is federated identity. This takes the distributed nature of OpenStack and allows it to span across multiple clouds, even from different providers. Two cloud providers can set up a trust relationship, enabling users of one provider to use the same credentials with another, trusted provider. Thus the same workload management tools you use for a single cloud can theoretically be used to manage workloads across multiple clouds. For capacity burst use cases, this is a powerful feature.

OpenStack Distributions

With the complexity of the architecture, a number of companies have stepped in to help with installation and management of an OpenStack platform in a private cloud. These include names familiar from the Linux distribution world, such as RedHat, SUSE, and Canonical (Ubuntu), as well as new players that are focused only on OpenStack, such as Mirantis.

INDUSTRY CONSOLIDATION

In fact the OpenStack industry has seen a great deal of consolidation in 2014 and 2015. Several pure-play OpenStack companies have been gobbled up by the bigger players.

Many large integrators and enterprise software vendors are also jumping into the OpenStack distribution game, with the likes of IBM, HP, and Oracle joining the fray.

If you don't have an OpenStack cloud available already, or you want to learn more about the architecture and how all of the pieces fit together, you can setup your own OpenStack playground. You can use one of these distributions to setup your own small cloud. Each of the distribution vendors provides their own tools for setup of OpenStack. They are primarily targeted at production environments, and as such can be pretty hard to get started with on your own. For example, Canonical's offering requires a minimum of seven physical nodes just to bring up the environment.

If you are setting something up small, your best options are probably RedHat's open source distribution (as opposed to their supported version running on RedHat Enterprise Linux), called RDO (`www.rdoproject.org`). The nice thing about this distribution is that it offers a simple "all in one" option to deploy the entire environment on a single node.

If you would like to tinker with the actual code of the various OpenStack services, you could also setup a *devstack* environment. Devstack (`www.devstack.org`) is a powerful set of scripts to create and configure an OpenStack development environment.

While the detailed instructions online are quite good, here are a few hints to make your devstack setup go smoothly. You'll want a fresh Ubuntu (`http://www.ubuntu.com`) or Fedora (`www.fedoraproject.org`) installation. Don't try to run devstack on your regular machine—you'll want a dedicated machine (virtual or physical). If you have a virtualization product like VMware Workstation or Fusion, or the free VirtualBox for your laptop or desktop, the best thing to do is create a base server installation of your OS of choice (enabling all of the extra repositories), and then snapshot it. This will make it easy to start over if you trash your environment.

The instructions will have you create a `local.conf` file, which the devstack scripts use to capture all of the specifics of your installation. There are only a few items you need to set in your local.conf.

```
[[local|localrc]]
ADMIN_PASSWORD=stack
DATABASE_PASSWORD=$ADMIN_PASSWORD
RABBIT_PASSWORD=$ADMIN_PASSWORD
SERVICE_PASSWORD=$ADMIN_PASSWORD
SERVICE_TOKEN=some-random-string
FIXED_RANGE=10.0.0.0/24
FLOATING_RANGE=192.168.20.0/25
PUBLIC_NETWORK_GATEWAY=192.168.20.1

LOGFILE=/opt/stack/logs/stack.log

disable_service n-net
enable_service neutron q-svc q-agt q-dhcp q-l3 q-meta
```

The first section here sets up the networking. You should pick a `FIXED_RANGE` that does not overlap your existing network. Your `FLOATING_RANGE` can correspond to an existing unused subnet on your network, with the `PUBLIC_NETWORK_GATEWAY` being the local default gateway on your subnet.

The `LOGFILE` setting simply helps you debug if your devstack does not come up properly, whereas the remainder of the file disables Nova networking and enables Neutron networking.

You will need access to either devstack or another OpenStack instance to follow the examples throughout this book.

GETTING THE OPENSTACK CLI CLIENTS

To follow along with the examples, you'll need access to a machine with the OpenStack clients installed. You can learn how to install the clients at `http://docs.openstack.org/cli-reference/content/`, which will include instructions for a variety of operating systems. The examples in this book will use Linux.

The easiest way to use these clients is to set the necessary authentication information in environment variables:

```
$ export OS_USERNAME=username OS_PASSWORD=password ↵
  OS_TENANT_NAME=tenant-name
$ export OS_AUTH_URL=http://keystone-ip:keystone-port/v2.0
```

This allows you to call the clients without passing those parameters:

```
$ openstack flavor list
+----+-----------+-------+------+-----------+-------+-----------+
| ID | Name      |   RAM | Disk | Ephemeral | VCPUs | Is Public |
+----+-----------+-------+------+-----------+-------+-----------+
| 1  | m1.tiny   |   512 |    1 |         0 |     1 | True      |
| 2  | m1.small  |  2048 |   20 |         0 |     1 | True      |
| 3  | m1.medium |  4096 |   40 |         0 |     2 | True      |
| 4  | m1.large  |  8192 |   80 |         0 |     4 | True      |
| 42 | m1.nano   |    64 |    0 |         0 |     1 | True      |
| 5  | m1.xlarge | 16384 |  160 |         0 |     8 | True      |
| 84 | m1.micro  |   128 |    0 |         0 |     1 | True      |
+----+-----------+-------+------+-----------+-------+-----------+
```

If your services endpoints are using HTTPS, you'll need to change the *`OS_AUTH_URL`* to reflect that. If you are using self-signed certificates, you also need to pass in the `–insecure` option.

SUMMARY

In this chapter, you have learned about the various types of cloud computing—IaaS, PaaS, and SaaS—and how they related to one another. OpenStack fills the IaaS, and perhaps in the future the PaaS functions, in the clouds. More importantly you learned that driving costs lower while delivering more features, more quickly is the driving force behind the cloud computing revolution. Finally, you learned about the major components of OpenStack—Nova, Neutron, Glance, and Keystone, and how to set up a playground for experimenting with OpenStack.